Autonomous Robots (2023) 47:981–997
https://doi.org/10.1007/s10514-023-10133-5

Integrating action knowledge and LLMs for task planning and situation handling in open worlds

Yan Ding[1] · Xiaohan Zhang[1] · Saeid Amiri[1] · Nieqing Cao[1] · Hao Yang[2] · Andy Kaminski[2] · Chad Esselink[2] · Shiqi Zhang[1]

Received: 2 May 2023 / Accepted: 3 August 2023 / Published online: 29 August 2023

Abstract
Task planning systems have been developed to help robots use human knowledge (about actions) to complete long-horizon tasks. Most of them have been developed for "closed worlds" while assuming the robot is provided with complete world knowledge. However, the real world is generally open, and the robots frequently encounter unforeseen situations that can potentially break theplanner's completeness. Could we leverage the recent advances on pre-trained Large Language Models (LLMs) to enable classical planning systems to deal with novel situations? This paper introduces a novel framework, called COWP, for open-world task planning and situation handling. COWP dynamically augments the robot's action knowledge, including the preconditions and effects of actions, with task-oriented commonsense knowledge. COWP embraces the openness from LLMs, and is grounded to specific domains via action knowledge. For systematic evaluations, we collected a dataset that includes 1085 execution-time situations. Each situation corresponds to a state instance wherein a robot is potentially unable to complete a task using a solution that normally works. Experimental results show that our approach outperforms competitive baselines from the literature in the success rate of service tasks. Additionally, we have demonstrated COWP using a mobile manipulator. Supplementary materials are available at: https://cowplanning.github.io/

Keywords Task planning · Large Language Models · Situation handling · Open worlds

1 Introduction

Robots that operate in the real world frequently encounter long-horizon tasks that require multiple actions. Automated task planning algorithms have been developed to help robots sequence actions to accomplish those tasks (Ghallab et al., 2016). Closed world assumption (CWA) is a presumption that was developed by the knowledge representation community and states that "statements that are true are also known to be true" (Reiter, 1981). Most current task planners have been developed for closed worlds, assuming complete domain knowledge is provided and one can enumerate all possible world states (Knoblock et al., 1991; Hoffmann, 2001; Nau et al., 2003; Helmert, 2006). However, the real world is "open" by nature, and unforeseen situations are common in practice (Hanheide et al., 2017). As a consequence, current automated task planners that are largely knowledge-based tend to be fragile in open worlds rife with unforeseen situations. Figure 1 shows an example situation: *Aiming to grasp a cup for drinking water, a robot found that the cup was not empty*. Although one can name many such situations, it is

✉ Yan Ding
yding25@binghamton.edu

Xiaohan Zhang
xzhan244@binghamton.edu

Saeid Amiri
samiri1@binghamton.edu

Nieqing Cao
ncao1@binghamton.edu

Hao Yang
howieyang@gmail.com

Andy Kaminski
kaminski.andy@gmail.com

Chad Esselink
cesselink@outlook.com

Shiqi Zhang
zhangs@binghamton.edu

[1] The State University of New York at Binghamton, Binghamton, NY 13902, USA

[2] Ford Motor Company, Dearborn, MI 18900, USA

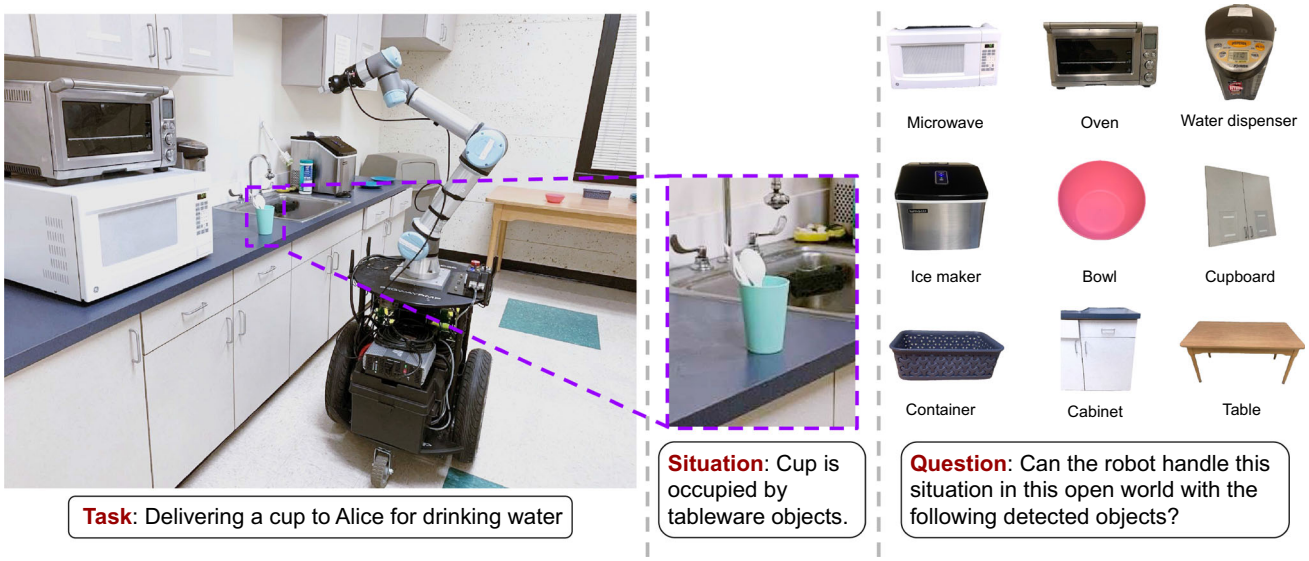

Fig. 1 An illustrative example of a *situation* in the real world, encountered during the execution of the plan "delivering a cup to a human for drinking water." The robot approached a cabinet in a kitchen room on which a cup was located. The robot then found the cup to be delivered. Before grasping it, however, the robot detected a situation that *the cup was occupied with a fork, a knife, and a spoon*. This situation prevented the robot from performing the current action (i.e., grasping) and rendered normal solutions for drinking water invalid. On the left is a kitchen environment and our mobile manipulator detecting a cup. On the right, we show a few visually perceived objects that were potentially useful for situation handling

impossible to provide a complete list of them. To this end, researchers have developed open-world planning methods for robust task completions in real-world scenarios (Jiang et al., 2019; Chernova et al., 2020; Hanheide et al., 2017; Kant et al., 2022; Huang et al., 2022; Brohan et al., 2023a, b; Zhang et al., 2023).

In the current literature, there are at least three ways of addressing the open-world planning problem. The first involves acquiring knowledge via human-robot interaction, e.g., dialog-based, to handle situations in an open-world context (Perera et al., 2015; Amiri et al., 2019; Tucker et al., 2020). Those methods require human involvement, which might hinder their autonomy capability and limit their applicability in the real world. A second idea for open-world planning relies on dynamically building a knowledge base to assist a pre-defined task planner, where this knowledge base is usually constructed in an automatic way using external information, e.g., using open-source knowledge graphs (Jiang et al., 2019; Chernova et al., 2020; Hanheide et al., 2017). Such external knowledge bases are considered "bounded" in this paper due to their representation and knowledge source, which limits the "openness" of their task planners. Large Language Models (LLMs) have been developed in recent years (Brown et al., 2020; Zhang et al., 2022; OpenAI, 2023; Google, 2023), and those LLMs have demonstrated major improvements in a variety of downstream tasks. Researchers have investigated the idea of extracting common sense from LLMs to guide robot task planning (Brohan et al., 2023a; Huang et al., 2022; Elsweiler et al., 2022; Kant et al., 2022). One challenge in this process is that the knowledge from LLMs is *domain-independent* (Davis & Marcus, 2015; Huang et al., 2023), whereas the robot faces specific domains that are featured with many *domain-dependent* constraints. For example, an LLM may provide a robot with the knowledge of how to serve water to people, but it falls short in determining the available types of containers (such as cups or glasses) the robot has access to, as well as the water sources in stock (like tap water or bottled water). In line with the third idea, we use LLMs for open-world planning in this paper. To address the challenge of grounding common sense to specific domains, we propose to enable the marriage of classical AI planning methods, and LLM-based knowledge extraction.

In this paper, we develop a robot task planning and situation handling framework, called *Common sense-based Open-World Planning* (**COWP**), that uses an LLM for dynamically augmenting automated task planners with external task-oriented common sense. COWP is based on classical planning and leverages LLMs to augment action knowledge (action preconditions and effects) for task planning and situation handling. The **main contribution** of this work is a novel integration of a pre-trained LLM with a knowledge-based task planner. Inheriting the desirable features from both sides, COWP is well grounded in specific domains while embracing commonsense solutions at large.

To conduct systematic evaluations, we selected 12 dining-focused tasks from the *ActivityPrograms* dataset (Puig et al.,

2018). Each task consisted of a high-level name, such as "Serve water", and a natural language description of the action sequence required to complete it. Using a crowdsourcing platform, we recruited 112 participants to propose potential situations that might hinder successful task execution. We gathered a situation dataset that includes more than one thousand situations for evaluation purposes. According to experimental results, we see COWP performed better than literature-selected baselines (Jiang et al., 2019; Huang et al., 2022; Singh et al., 2023) in terms of the respective success rates in task completion and situation handling. We implemented and demonstrated COWP using a mobile manipulator.

2 Background and related work

In this section, we first briefly discuss classical task planning methods that are mostly developed under the closed world assumption. We then summarize three families of open-world task planning methods for robots, which are grouped based on how unforeseen situations are addressed.

2.1 Classical task planning for closed worlds

A classical task planning problem consists of two main components: a domain description and a problem description (Haslum et al., 2019). The domain description includes a collection of actions, each of which is defined by its preconditions and subsequent effects. The problem description, on the other hand, specifies the initial state and the desired goal conditions. A sequence of actions can be generated, enabling an effective transition from the initial state to the designated goal state.

Closed world assumption (CWA) indicates that an agent is provided with complete domain knowledge, and that all statements that are true are known to be true by the agent (Reiter, 1981). In this paper, such a classical planning system is referred to as a closed-world task planner. Although robots face the real world that is open by nature, their planning systems are frequently constructed under the CWA (Hanheide et al., 2017; Jiang et al., 2019; Galindo et al., 2008; Ghallab et al., 2016; Haslum et al., 2019; Nau et al., 2003). The consequence is that those robot planning systems are not robust to unforeseen situations at execution time. In this paper, we aim to develop a task planner that is aware of and able to handle unforeseen situations in open-world scenarios.

2.2 Open-world task planning with human in the loop

Task planning systems have been developed to acquire knowledge via human-robot interaction to handle open-world situations (Perera et al., 2015; Amiri et al., 2019; Tucker et al., 2020). For instance, researchers created a planning system that uses dialog systems to augment their knowledge bases (Perera et al., 2015), whereas Amiri et al. (2019) further modeled the noise in language understanding (Amiri et al., 2019). Tucker et al. (2020) enabled a mobile robot to ground new concepts using visual-linguistic observations, e.g., to ground the new word "box" given command of "move to the box" by exploring the environment and hypothesizing potential new objects from natural language (Tucker et al., 2020). The major difference from those open-world planning methods is that COWP does not require human involvement.

2.3 Open-world task planning with external knowledge

Some existing planning systems address unforeseen situations by dynamically constructing an external knowledge base for open-world reasoning. For instance, researchers have developed object-centric planning algorithms that maintain a database about objects and introduce new object concepts and their properties (e.g., location) into their task planners (Jiang et al., 2019; Chernova et al., 2020). For example, Jiang et al. Jiang et al. (2019) developed an object-centric, open-world planning system that dynamically introduces new object concepts through augmenting a local knowledge base with external information (Jiang et al., 2019). In the work of Hanheide et al. Hanheide et al. (2017), additional action effects and assumptive actions were modeled as an external knowledge to explain the failure of task completion and compute plans in open worlds (Hanheide et al., 2017). A major difference from their methods is that COWP employs an LLM as a generative approach that is capable of responding to any situation, whereas the external knowledge sources of those methods limits the openness of their systems.

2.4 Closed-world task planning with LLMs

A straightforward idea for LLM-based planning is to directly enter planning domain information into LLMs, and request plan generations. Recent research has shown that a naive implementation of such ideas produces very poor performance even if the planning domain is as simple as Blocksworld (Valmeekam et al., 2022, 2023). The ChatGPT report (in its conclusion section) also identified "long-horizon planning" as a challenging task, and encouraged more research on LLM-based planning (OpenAI, 2023). Very recently, researchers have investigated translating descriptions of planning tasks in natural language into PDDL (Haslum et al., 2019), and then computing plans using PDDL systems (Liu et al., 2023). The produced system called LLM+P

takes natural language descriptions as input, and computes plans with optimality guarantee. The action knowledge of LLM+P formulated in PDDL was provided by domain experts, and could not adapt to novel situations at execution time. By comparison, COWP (ours) dynamically extract common sense from LLMs for augmenting its action knowledge for planning and situation handling.

2.5 Open-world task planning with LLMs

In recent years, many LLMs have emerged, including BERT (Devlin et al., 2019), GPT-3 (Brown et al., 2020), ChatGPT (OpenAI, 2023), CodeX (Chen et al., 2021), and OPT (Zhang et al., 2022). These LLMs encode a large amount of commonsense knowledge (Liu et al., 2023; Wang et al., 2021; Li et al., 2003; West et al., 2022) and have been employed in robot task planning (Kant et al., 2022; Huang et al., 2022; Brohan et al., 2023a; Huang et al., 2022; Singh et al., 2023; Ding et al., 2023). For instance, the work of Huang et. al. (Huang et al., 2022) showed that LLMs can effectively facilitate task planning in household environments by iteratively augmenting prompts (Huang et al., 2022). The SayCan system enabled robot planning with affordance functions to account for action feasibility, where the service requests are specified in natural language (e.g., "deliver a Coke") (Brohan et al., 2023a). Additionally, various teams also have successfully applied LLMs to generate plans for high-level tasks expressed in natural language by sequencing actions (Huang et al., 2022; Kant et al., 2022; Xie et al., 2023; Song et al., 2022; Lin et al., 2023; Ding et al., 2023).

It is generally difficult to ground LLM-based planning systems in the real world as the LLMs were not trained for specific domains. For instance, LLM-based planners frequently generate plans that require objects or tools that are not present in the scene (Huang et al., 2022). COWP (ours) alleviates this issue through reasoning about robot skills using rule-based action knowledge.

Work closest to COWP is a recent LLM-based planning system, called ProgPrompt (Singh et al., 2023), which generates task plans and handles situations using programmatic LLM prompts. ProgPrompt handles situations by asserting preconditions of the plan (e.g., being close to the fridge before attempting to open it) and responding to failed assertions with appropriate recovery actions. Compared with ProgPrompt, which relies on example solutions in prompting to guide the LLMs, COWP (ours) uses action knowledge to enable zero-shot prompting for planning and situation handling. For example, in the case of ProgPrompt, three examples are typically provided in the prompt. These examples guide ProgPromptto generate task plans, and these plans take into account feedback from the environment by including preconditions for actions. By comparison, COWP makes use of action knowledge, which can be in the form of rules, facts, and principles related to the task. This can include a detailed understanding of how actions impact the state of the environment, which is not required in the prompts used by ProgPrompt. This enables COWP to create task plans without the need for example solutions, i.e., zero-shot prompting.

COWP extracts commonsense knowledge from LLMs and incorporates rule-based action knowledge from human experts. Reasoning with action knowledge ensures the soundness of task plans generated by COWP, while querying LLMs guarantees the openness of COWP to unforeseen situations. As a result, COWP can be better grounded to specific domains, and is able to incorporate common sense to augment robot capabilities supported by predefined skills.

3 Algorithm

In this section, we first provide a problem statement and then present our open-world planning approach called Common sense-based Open-World Planning (COWP).

Our goal is to address open-world planning problems for robots. An open-world planning problem assumes that the robot might encounter situations that are not considered in the development of the planning systems. More specifically, we use the term of *situation* to refer to an unforeseen world state that potentially prevents an agent from completing a task using a solution that normally works. In this paper, we assumed the availability of a description of the robot's skills, formulated in action description language PDDL. PDDL is designed to formalize AI planning problems, allowing for a more direct comparison of planning algorithms and implementations (Aeronautiques et al., 1998). The *objective* of an open-world planner is to compute plans that can adapt to and handle unexpected situations while pursuing the completion of service tasks or reporting "no solution" when appropriate.

3.1 Algorithm description

Fig. 2 illustrates the three major components of our COWP framework. **Task Planner** is used for computing a plan under the closed-world assumption and is provided as prior knowledge in this work. **Plan Monitor** evaluates the overall feasibility of the current plan using common sense. **Knowledge Acquirer** is for acquiring common sense to augment the robot's action effects when the task planner generates no plan.

Algorithm 1 describes how the components of COWP interact with each other. Initially, Task Planner generates a satisfying plan based on the goal provided by a human user in **Line** 2. After that, the actions in the plan are performed sequentially by a robot in the for-loop of **Lines** 3–20. If the current plan remains feasible, as evaluated by Plan Monitor,

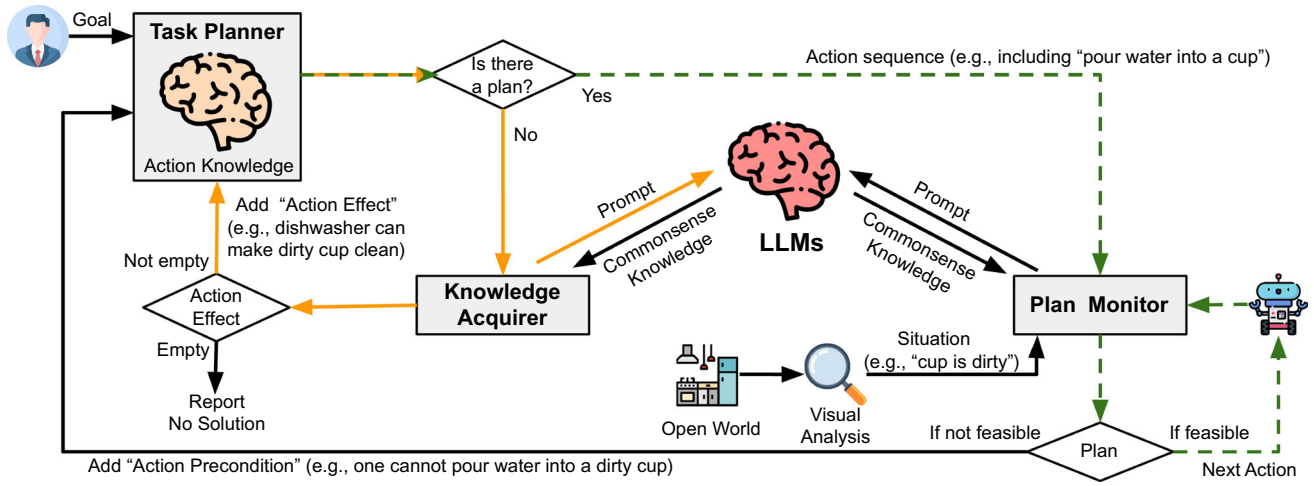

Fig. 2 An overview of **COWP** that includes the three key components of Task Planner (provided as prior knowledge under closed-world assumption), Knowledge Acquirer, and Plan Monitor. The **green** (dashed) loop represents a plan execution process where the robot does not encounter any situation, or these situations have no impact on the robot's plan execution. The **orange** loop is activated when the robot's current (closed-world) task planner is unable to develop a plan, which activates Knowledge Acquirer to augment the task planner with additional action effects utilizing common sense (Color figure online)

Algorithm 1: COWP algorithm

```
Require: Task Planner, Plan Monitor, Knowledge Acquirer, and
         an LLM
Input: A domain description, and a problem description
1  while task is not completed do
2      Compute a plan pln using Task Planner given the domain
       description and the problem description;
3      for each action in pln do
4          Evaluate the feasibility of the current plan using Plan
           Monitor given a situation;
5          if a plan is not feasible then
6              Add an action precondition to Task Planner;
7              Compute a new task plan pln', and pln ← pln';
8              if pln is empty then
9                  Extract common sense (action effects) using
                   Knowledge Acquirer;
10                 if action effect is not empty then
11                     Add an action effect to Task Planner;
12                     Compute a new task plan pln'', and pln ←
                       pln'';
13                 else
14                     Report "no solution";
15                 end
16             end
17         else
18             Execute the next action by the robot;
19         end
20     end
21 end
```

the next action will be directly passed to the robot for execution in **Line** 18; otherwise, an action precondition will be added to Task Planner in **Line** 6. For example, when Task Planner does not know anything about "dirty cups," it might wrongly believe that one can use a dirty cup as a container for drinking water. In this situation, **Line** 6 will add a new statement "*The precondition of filling a cup with water is that the cup is not dirty*" into the task planner. After the domain description of Task Planner is updated (adding action preconditions), the planner tries to generate a new plan pln' in **Line** 7. If no plan is generated by Task Planner (**Line** 8), Knowledge Acquirer will be activated for knowledge augmentation with external task-oriented common sense in **Line** 9. If the extracted common sense includes action effects, such information will be added into the task planner in **Line** 11. For instance, the robot might learn that a chopping board can be used for holding steak, as an action effect that was unknown before. This process continues until no additional action effects can be generated (in this case, COWP reports "no solution" in **Line** 14 or a plan is generated.

COWP leverages common sense to augment a knowledge-based task planner to address situations towards robust task completion in open worlds. The implementations of **Lines** 4 and 9 using common sense are non-trivial in practice. Next, we describe how commonsense knowledge is extracted from an LLM for those purposes.

3.2 Plan monitor and knowledge acquirer

In this section, two important components of COWP are discussed, i.e., Plan Monitor and Knowledge Acquirer.

Plan Monitor is designed to evaluate if there are any situations that could prevent a robot from completing its current task successfully. To achieve this, the plan monitor takes in action sequences generated from a classical planner and a set

of situations collected from the real world. The Plan Monitor uses a prompt template to query an LLM for each action in the sequence. The prompt template is based on the following structure:

Prompt 1 *Is it suitable for a robot to [Perform-Action], if [Situation]?*

The placeholder [] represents the specific information to be filled in within the template. In this case, *[Perform-Action]* represents a natural language description of an action generated from our PDDL-based task planner, with the action in the form of "Action Object-1 Object-2...". *[Situation]* represents a natural language description of a situation. For example, if the current action is "Fill Cup Water" and the given situation is "Cup is broken", the corresponding prompt would be "*Is it suitable for a robot to fill a cup with water, if the cup is broken?*". Translating a symbolic action into a natural language description can be achieved either by following handcrafted rules or with the assistance of LLM's grammar completion. In this case, we adopt the first approach. An LLM generates a natural language response to each prompt. If the LLM determines that the action is infeasible given the situation described, the whole plan is considered infeasible.

3.2.1 Knowledge acquirer

Aims to acquire common sense for augmenting the task planner's action knowledge for situation handling. In particular, the input for the knowledge acquirer consists of a collection of available objects in the environment that the robot can access. Another template-based prompt is developed for querying an LLM for acquiring common sense about action effects.

Prompt 2 *Is it suitable for a robot to [Perform-Action-with-Object]?*[1]

In the prompt, the placeholder *[Perform-Action-with-Object]* represents a natural language description of a robot performing an action with another object. For example, if the action "Fill Cup Water" is considered infeasible under the situation "Cup is broken" by Plan Monitor, one instance of using Prompt 2 with another object "bowl" is "*Is it suitable for a robot to fill a bowl with water?*". If the response from an LLM to the example prompt is "*Yes*", an additional action effect, "The bowl can also be a container to fill water", will be added to the task planner.

Continuing the "Serve water" example, additional action effects introduced through Prompt 2 might enable the task planner to generate many feasible plans, e.g., using a glass, measuring cup, or bowl to fill water. It can be difficult for the task planner to evaluate which plan makes the best sense. To this end, we develop Prompt 3 for selecting a task plan of the highest quality among those satisficing plans:

Prompt 3 *There are some objects, such as [Object-1], [Object-2],..., and [Object-N]. Which is the most suitable for [Current-Task], if [Situation]?*

The placeholder *[Current-Task]* is a high-level task description in natural language. In the example of "Serve water", we expect the LLM to respond by suggesting "glass" being the most suitable for holding water among those mentioned items.

3.3 Implementation

Here, we explain how to implement COWP using the "Serve water" task as an example. To do so, we use a PDDL-based closed-world task planner, which requires both a domain file and a problem file. The domain file defines a set of predicates (e.g., `is_grasped`) and actions (e.g., *fill*), with each action specified by its preconditions and effects. Consider the following action definition for *fill* shown in Fig. 3, which includes preconditions like `(is_grasped ?c)` ∧ (is_empty ?c) and effects such as `(is_filled ?c)`:

The problem file, on the other hand, defines the task by specifying an initial state and a goal state, which in this case is "Water is served to the user". To generate a task plan, a solver, such as Fast Downward (Helmert, 2006) can be used. A feasible closed-world plan for this task is shown in Fig. 4.

```
(:action fill
        :parameters
        (?r - robot ?c - cup
         ?f - faucet ?l - location)
        :precondition
        (and (is_grasped ?c)
             (is_empty ?c)
             (faucet_at ?f ?l)
             (is_on ?f)
             (robot_at ?r ?l))
        :effect
        (and (is_filled ?c)
             (not (is_on ?f))
             (is_off ?f))
)
```

Fig. 3 Definition of action "*fill*" in our PDDL-based task planner

[1] Intuitively, the solution from COWP goes beyond finding alternative objects in addressing unforeseen situations. COWP enables situation handling by manipulating the attributes of individual instances. For example, a "dirty cup" situation can be handled by running a dishwasher, where no second object is involved.

```
S1: find robot cup kitchen
S2: find_faucet robot faucet kitchen
S3: turnon robot faucet kitchen
S4: grasp robot cup kitchen
S5: fill robot cup faucet kitchen
S6: move robot cup kitchen dining
S7: place robot cup table dining
```

Fig. 4 One closed-world plan for the task "Serve water"

```
(:action fill
        :parameters
        (?r - robot ?c - cup
         ?f - faucet ?l - location)
        :precondition
        (and (is_grasped ?c)
             (is_empty ?c)
             (not (is_dirty ?c))
             (faucet_at ?f ?l)
             (is_on ?f)
             (robot_at ?r ?l))
        :effect
        (and (is_filled ?c)
             (not (is_on ?f))
             (is_off ?f))
)
```

Fig. 5 An action constraint, `not (is_dirty ?c)`, is added into the action "*fill*"

Plan Monitor is responsible for checking action feasibility under a given situation, such as "Cup is dirty". If an action, like *fill*, is considered infeasible by an LLM, a constraint, such as `not (is_dirty ?c)`, will be added to the action precondition. We implement a predicate generator that translates the natural language situation into a symbolic form, such as `is_dirty ?cup`. This generator utilizes the few-shot learning capabilities of the LLM. Once the predicate is obtained, it is added to the PDDL code and highlighted with an underline, as shown in Fig. 5. Additionally, the initial state in the problem file is updated to reflect the new situation, `(is_dirty cup)`.

After adding the constraint `not (is_dirty ?c)`, the planning system might not generate a feasible plan, as the object *cup* is found to be dirty and cannot be used. In such cases, the planning system needs to seek alternative solutions to accomplish the task by *adding action effects* to the planner. The commonsense knowledge that "Bowl can be a container to fill water" is obtained by Knowledge Acquirer. This knowledge might have been overlooked by the planning system developer. COWP can add this new action effect into the planning system, as shown in Fig. 6. Now, the *fill* action can be applied to both *cup* and *bowl*. Additionally, related parts of the code are adjusted accordingly, and the initial state in the problem file is updated to include the bowl's location.

```
(:action fill
        :parameters
        (?r - robot ?b - bowl
         ?f - faucet ?l - location)
        :precondition
        (and (is_grasped ?b)
             (is_empty ?b)
             (faucet_at ?f ?l)
             (is_on ?f)
             (robot_at ?r ?l))
        :effect
        (and (is_filled ?b)
             (not (is_on ?f))
             (is_off ?f))
)
```

Fig. 6 The PDDL-based task planner incorporates the commonsense knowledge that "Bowl can be a container to fill water" as an action effect

```
S1: find robot bowl kitchen
S2: find_faucet robot faucet kitchen
S3: turnon robot faucet kitchen
S4: grasp robot bowl kitchen
S5: fill robot bowl faucet kitchen
S6: move robot bowl kitchen dining
S7: place robot bowl table dining
```

Fig. 7 A feasible plan for the task "Serve water", which can complete the service task "Serve water" and handle the situation "Cup is dirty."

An alternative plan that uses a bowl for serving water, shown in Fig. 7, can complete the service task "Serve water" and handle the situation "Cup is dirty".

4 Experiments

In this section, we evaluate COWP's performance in planning and situation handling.

4.1 Experimental setup

Our experiments were performed in a *dining domain*, where a service robot is tasked with fulfilling a user's service requests in a home setting. For simulating dining domains, we chose 12 everyday tasks from the *ActivityPrograms* dataset (Puig et al., 2018). These tasks can be found in Table 1. For each task, we constructed PDDL-based planning systems to generate action sequences for their completion.

To carry out our experiments, we used OpenAI's GPT-3 engines. Please refer to Table 2 for the specific hyperparameters we adopted. In simulation experiments, we assume that our robot is provided with a perception module for converting raw sensory data into logical facts (situations), such as objects and their properties, while the robot still needs to reason about the facts for planning and situation handling. Our

Table 1 12 tasks extracted from the ActivityPrograms dataset (Puig et al., 2018) for evaluation purposes

Task name	Task name	Task name
Set table	Serve water	Serve coke
Wash plate	Heat burger	Make coffee
Clean floor	Prepare burger	Store food
Wash cup	Wash sink	Wash glass

Table 2 Hypermeters of OpenAI's GPT-3 engines in Our Experiment

Parameter	Value
Model	Text-davinci-003
Temperature	0.0
Top p	1.0
Maximum length	32
Frequency penalty	0.0
Presence penalty	0.0

experiments were performed using simulated robot behaviors and situations collected from human participants.

4.2 Simulation platform

4.2.1 Situation dataset

To evaluate the performance of COWP in dealing with situations in a dining domain, we collected a dataset of execution-time situations using Amazon Mechanical Turk. Each instance of the dataset corresponds to a situation that prevents a service robot from completing a task. We recruited MTurkers with a minimum HIT score of 70. Each MTurker was provided with a task description, including steps for completing the task. The MTurkers were asked to respond to a questionnaire by identifying one step in the provided plan and describing a situation that might occur in that step. For example, in the task "Serve water", we provided its plan consisting of the following steps: "1) Walk to kitchen room, 2) Find glass, 3) Find sink, 4) Find faucet, 5) Turn on faucet, 6) Fill glass with water, 7) Move glass near table, 8) Place cup on table". Two common responses from the MTurkers were "*Glass is broken*" for Step 2 and "*Faucet has no water*" for Step 5.

We received 1224 responses from MTurkers, and 1,085 of them were valid, where the responses were evaluated through both a validation question and a manual filtering process. We included a validation question that resembled other questions in the middle of the questionnaire, but instead of asking for a situation, it required the MTurker to input a provided "secret code", such as "Successfully Verified!". Regarding the manual filtering process, we checked the responses for relevance to the task and logical coherence. For instance, some MTurkers copied and pasted text that was completely irrelevant, and those responses were removed manually. Additionally, while some responses like "Glass's color is green" were related to the task, such as "Serve water," they were considered invalid because the robot can still complete the task regardless of the glass's color. There are at least 88 situations collected for each of the 12 tasks. To facilitate the construction of the simulation platform, we further grouped the situations of significant similarities and generate a set of "distinguishable" situations. For instance, two situations, "Glass is broken" and "Glass is shattered" are considered indistinguishable. As a result, each task has 12 to 24 distinguishable situations, and the specific number of distinguishable situations for each task can be found in Table 3. Here, we show 15 distinguishable situations for the task "Serve water":

(1) *Glass is broken.*
(2) *Faucet has no water.*
(3) *Glass is dusty.*
(4) *Glass is missing.*
(5) *Water is dirty.*
(6) *Faucet has sustained physical damage.*
(7) *Faucet cannot be turned on.*
(8) *Sink is not found.*
(9) *Faucet is leaking.*
(10) *Faucet is not found.*
(11) *Water spills on floor.*
(12) *Glass is not full of water.*
(13) *Kitchen door is locked and cannot be opened.*
14) *Glass falls onto floor.*
15) *Glass is stuck and cannot be removed.*

On our project website (https://cowplanning.github.io/), users can download both the MTurk questionnaire and the situation dataset. The dataset is provided as a CSV file comprising 12 separate sheets, each representing the situations for a distinct task. The name of the task corresponds to the sheet name. Each sheet consists of five columns. The situations' descriptions provided by the MTurkers are in Column A. Column B details the corresponding steps where the described situation occurs. Column C is the index of distinguishable situations, while Column D provides descriptions of these situations. Finally, Column E indicates the number of distinguishable situations.

Note that *this dataset is intended solely for evaluation purposes*. The LLMs utilized in COWP act as zero-shot learners, and Plan Monitor can successfully perform its intended function without situation examples.

Table 3 The number of distinguishable situations for each task

Task name	Num	Task name	Num
Make coffee	24	Set table	16
prepare burger	21	Clean floor	15
Heat burger	19	Serve water	15
Wash sink	18	Wash cup	15
Store food	17	Serve coke	14
Wash plate	16	Wash glass	12

4.2.2 Simulator

We have constructed a simulator capable of generating a high-level task name (e.g., "Serve water") and an unexpected situation (e.g., "Glass is broken") that occurs during one of the steps (e.g., "Fill glass with water") in each trial. Here, we assume that the probability of the robot encountering a situation while executing an action is denoted by P (i.e., 0.1). This assumption implies that a longer task plan may be more likely to encounter such situations. The simulator also contains 86 objects, including cups, burgers, forks, tables, and chairs. In each trial, it randomly selects and spawns half of the available objects for the robot to manipulate in order to resolve the situation. This setting helps create a diverse environment for the robot. These objects are also extracted from the ActivityPrograms dataset (Puig et al., 2018) and can be classified into five categories: kitchenware, appliance, furniture, food, and drink. The "kitchenware" category comprises the highest number of objects (29), while the "drink" category contains the fewest (8). For more details, please visit our website at https://cowplanning.github.io/.

4.2.3 Baselines and evaluation metrics

The evaluation of open-world planners is based on the respective *success rates* of a robot completing service tasks and handling situations in a dining domain. The following five baselines have been used in our experiments:

- Inner Monologue (Huang et al., 2022) leverages environmental feedback to generate task plans and handle situations. In the original implementation, this approach could process three types of textual feedback: Passive Scene Description, Success Detection, and Active Scene Description. Passive Scene Description involves obtaining feedback, such as details regarding available objects and situational context, from the environment. One instance of this feedback could be "cup, the cup is broken, and drinking glass." Success Detection checks if every action in the generated plan is successfully executed. However, we have deactivated Active Scene Description because our system excludes humans from the loop. The specific prompts used in Inner Monologue are available in the supplementary material on our project website.
- Closed World (CW) corresponds to classical task planning developed for closed-world scenarios. In practice, CW was implemented by repeatedly activating the closed-world task planner and updating the current world state after executing each action. CW does not have the capability to handle situations. For example, CW is unable to generate a plan that involves using a bowl as a container for serving water.
- External Knowledge (EK) (Jiang et al., 2019) is a baseline approach that enables a closed-world task planner to acquire knowledge from an external source. In our implementation, this external source provides information about a half of the domain objects. For instance, EK may generate a plan that involves using a bowl as a container for serving water, if its knowledge base contains "Bowl can be used for serving water".
- Language Models as Zero-Shot Planners (LMZSP) (Huang et al., 2022) is a baseline that leverages LLM to compute task plans, where domain-specific action knowledge is not utilized.The LMZSP baseline (Huang et al., 2022) was not grounded. As a result, their generated plans frequently involve objects unavailable or inapplicable in the current environment. Furthermore, LMZSP cannot receive feedback from its environment, restricting its capability to handle situations. The hyperparameters used in LMZSP are available in the provided supplementary material on our project website.
- ProgPrompt (Singh et al., 2023) serves as the baseline method that utilizes a programmatic LLM prompt structure to facilitate open-world task planning. The method consists of two types of prompts, namely PROMPT for Planning and PROMPT for State Feedback. ProgPrompt is a few-shot learning method, unlike LMZSP. This isa *competitive* baseline. More specifically, ProgPrompt relies on the "PROMPT for Planning" to create a plan based on the task specification, including a contextual description of the situation. The plan takes feedback from the environment into account by including preconditions for actions. For instance, it might include situated state feedback like "assert ('dirty' to 'cup') else: find('drinkingglass')". If the state is "cup is dirty", the "find drinking glass" action will be executed. The 'State Feedback Prompt' uses feedback from the environment to trigger preconditions in the generated plan. In this example, a context description like "cup is dirty" would trigger the "assert ('dirty' to 'cup')" precondition. This is a competitive baseline. ProgPrompt has the same access to feedback inputs from the environment as our proposed approach, ensuring a fair comparison between the two

methods. For more detail on the exact prompts we used in our research with ProgPrompt, please refer to the provided supplementary materials on our project website.

Baselines ProgPrompt, Inner Monologue and LMZSP have utilized the GPT-3 in their experimental implementations. It is crucial to note that GPT-3 offers multiple models, including "*text-davinci-002*" or "*text-davinci-003*". However, the specific model used by these studies was not mentioned. To ensure a fair comparison, we have chosen to use the "*text-davinci-003*" model for all approaches.

4.2.4 Success criteria

Unlike many classical planning systems (Lo et al., 2020; Jiang et al., 2019; Garrett et al., 2021, 2020) that are guaranteed to provide sound solutions, LLM-based planning systems might generate invalid solutions without being aware of them. For instance, GPT-3 might suggest that one can use a pan for drinking water, which is technically possible, but very uncommon in our everyday life. In this paper, we intend to consider those to be unsuccessful trials. LLM-based task planners might wrongly believe that a good-quality plan is generated, while it is actually not the case. To compare with the ground truth, we recruited a second group of people, including six volunteers, to evaluate the performance of different open-world task planners. The volunteers included two females and four males aged 20–40, and all were graduate students in engineering fields. A successful task plan is defined as one that fulfills a user's service request through actions that the robot is capable of executing and are acceptable to humans. In addition, common factors that could render a plan unsuccessful include, but are not limited to, the inclusion of non-existent objects in the environment, actions that the robot is unable to execute, or missing essential steps between actions. Detailed instructions provided for the volunteers prior to the evaluation are available in the provided supplementary materials on our project website.

4.2.5 COWP versus baselines (overall)

Table 4 shows the overall performance of COWP in task planning and situation handling, as compared to five baselines, i.e., ProgPrompt, Inner Monologue, EK, LMZSP, and CW. The table demonstrates that COWP outperforms all five baselines in terms of task completion and situation handling percentages. However, LMZSP and CW show poor performance. We believe the poor performance of CW is caused by its inability to leverage external information to handle unforeseen situations. For LMZSP, its poor performance is caused by its weakness in grounding general commonsense knowledge in specific domains and the big noise in the generated plans. As a result, many "solutions" generated by LMZSP are not executable by the robot. For instance, in one plan, "Find cup; Grab cup; Walk to sink; Run to cup; Walk to water; Find water; Grab water; Pour water into cup" generated by LMZSP, some essential steps were omitted, such as "Walk to kitchen" and "Turn on faucet". Additionally, the robot was unable to execute the "Grab water" action. This limitation of LMZSP has also been observed and analyzed in the paper by Singh et al. (2023).

The performance of ProgPrompt is is not as good as that of COWP. This can be attributed to the fact that the task plans produced by ProgPrompt might include noise, such as logical flaws, which reduces the chance of successfully completing the task. For example, one plan generated by ProgPrompt includes the two consecutive actions of "fill glass with water" and "turn on faucet", which is invalid, because "turn on faucet" should be executed before "fill glass with water." Additionally, we find the ProgPrompt's capability relies on the provided examples in the prompt, while COWP (ours) handles situations via zero-shot prompting of LLM.

Inner Monologue's performance falls short when compared to COWP (ours) and ProgPrompt. Our observation indicates that Inner Monologue tends to generate incomplete task plans and exhibit a bias toward "Bounding Thinking," which consequently leads to a decrease in both task completion and situation handling percentages. Take, for instance, an Inner Monologue-produced task plan for serving water: "Step 0: walk to kitchen; Step 1: find drinking glass; Step 2: fill drinking glass with water; Step 3: find kitchen table; Step 4: put drinking glass on kitchen table; Step 5: done". This plan omits the crucial step of grasping the drinking glass first before filling it with water (Step 2), and overlooks the necessary action to access water, like "Switch on Faucet",

Table 4 The overall performances of COWP (ours) andfivebaseline methods are compared in terms of their task completion and situation handling percentages. The task completion percentage is calculated by dividing the number of completed tasks by the total number of trials, which is 150. The situation handling percentage is the ratio of the number of handled situations to the total number of situations

	COWP (ours)	ProgPrompt	Inner Monologue	EK	CW	LMZSP
Task completion percentage (%)	**67.8**	61.8	54.0	55.1	44.3	10.2
Situation handling percentage (%)	**36.9**	30.1	22.1	17.3	0.0	0.0

Bold values represent the best results obtained in the respective experiments and comparisons

given the environment does not have water readily available. This lack of completeness compromises the robot's ability to execute the task. Furthermore, Inner Monologue has been shown to be lacking in situation handling. Despite being informed about situations through Passive Scene Description - a method of gathering environmental context, it often disregards this crucial contextual information. For instance, Inner Monologue may recommend using a cup to serve water even if the "cup is dirty." Moreover, it struggles to identify suitable solutions to address the situation. In the same scenario where a dirty cup is present, Inner Monologue might suggest using a water boiler or a bucket to serve water to humans, even when more suitable options, such as a mug or a clean drinking glass, are available in the environment.

4.2.6 COWP versus baselines by task

Figure 8 shows the results of comparing COWP and five baselines in the success rate of task completion under *different tasks*. The x-axis denotes the task name, and the y-axis indicates the percentage of task completion. From the figure, we can see COWP outperforms the five baselines in all tasks. It is quite interesting to see that open-world planners (including COWP) work better in some tasks than the others. For instance, in the "Set table" task, there are many "missing object" situations, and it happened that the robot could easily find alternative tableware objects to address those situations with the assistance of GPT-3. However, some tasks such as "Wash glass" are more difficult because many situations are beyond the robot's capabilities, e.g., "There is a power outage" and "Faucet has no water".

Figure 9 shows the results of comparing COWP and three baselines in the success rate of situation handling under *different tasks*. In this case, we do not include the baseline CW into the figure, because it's inapplicable to the task of situation handling. The results indicate that COWP outperforms the baselines in all tasks. Specifically, COWP was able to handle over 45% of the situations in the first five tasks, and achieved the best performance of around 70% in the "Set table" task. There are some tasks where situation handling is more difficult for COWP. For instance, there are situations, such as "There is no water in the sink to wash the plate.", and

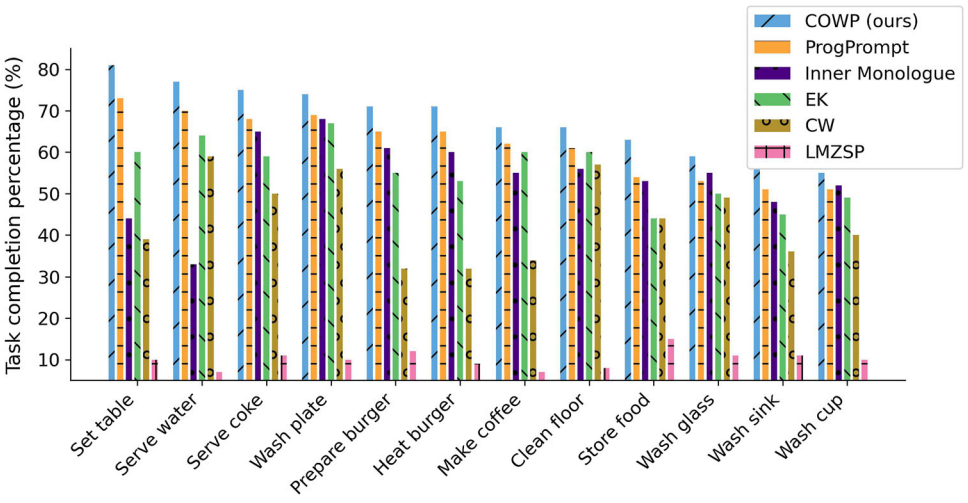

Fig. 8 The task completion percentage of COWP (ours) and five baseline methods under **12 different tasks**. The *x-axis* represents the task name, and the *y-axis* represents the task completion percentage. The task completion percentage for each value is an average of 150 trials. The tasks are sorted based on the performance of COWP, where the very left corresponds to its best performance

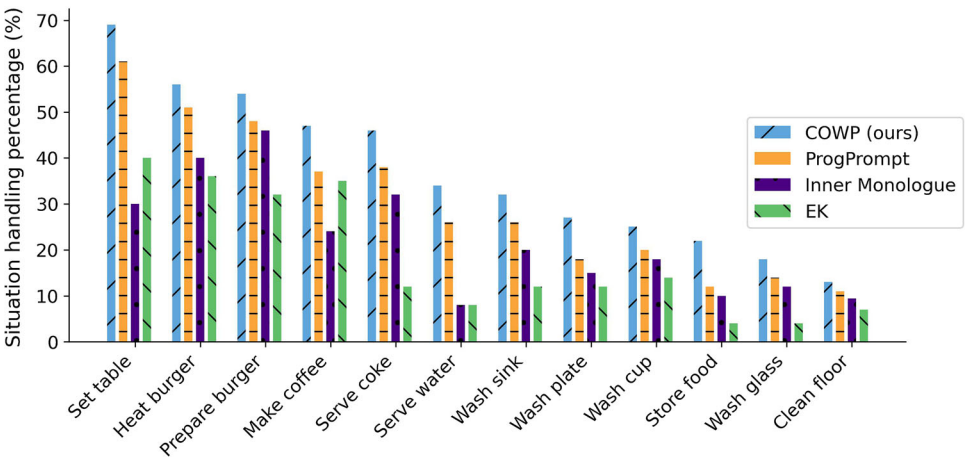

Fig. 9 The situation handling percentage of COWP (ours) and three baseline methods under **12 different tasks**, where the *x-axis* represents the task name, and the *y-axis* represents the situation handling percentage. Each y value represents a ratio of the number of handled situations to the total number of situations. The tasks are ranked based on the performance of COWP, where the very left corresponds to its best performance

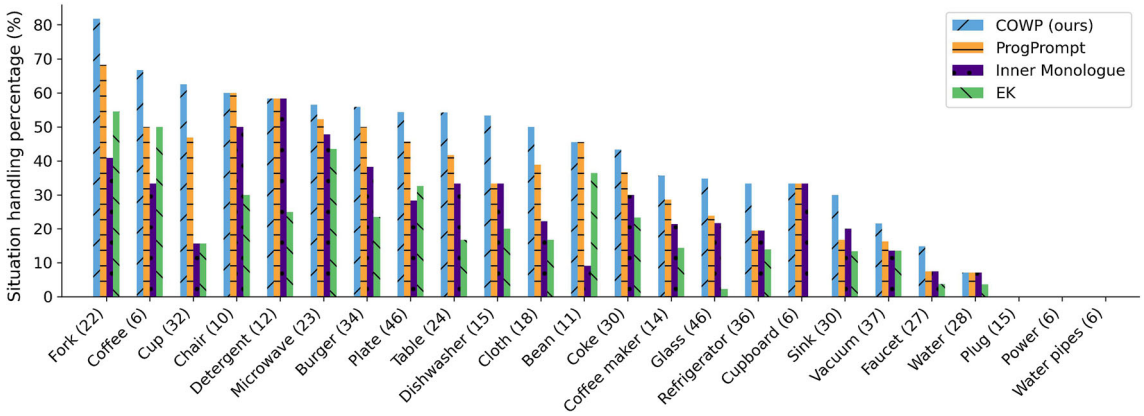

Fig. 10 The situation handling percentages of COWP (ours) and three baseline methods under **different objects**, where the *x-axis* represents the object involved in the sampled situation, the number (X) beside each object is the occurrence of the object in situations, and the *y-axis* represents the percentage of situation handling. The objects are ranked based on the performance of COWP. In this analysis, we only display objects with an occurrence of more than 5

"The robot cannot access the sink, as the door to the kitchen is locked." in task "Wash plate", where the robot could not do anything with its provided skills. By analyzing Figs. 8 and 9, we observe that the task "Wash plate" has a significantly higher task completion percentage than "Wash glass", despite having a similar situation handling percentage. This is due to "Wash plate" comprising only eight action sequences, compared to "Wash glass" with 12. Consequently, the robot is more prone to encountering situations in the "Wash glass" task, which hinders the robot's task completion rate.

4.2.7 COWP versus baselines by object

Figure 10 compares the performance of COWP and three baselines in situation handling percentage under *different objects*. The *x-axis* represents the objects in our situation dataset and their occurrences, and the *y-axis* represents the performance of planning methods in situation handling. For instance, common situations about cup include "dirty cup," "broken cup," and "missing cup." COWP (ours) performed the best over all objects, while some baselines produced comparable performances over some objects. An important observation is that COWP produced higher success rates in situations involving "fork", "coffee", "cup", and "chair". This is because many of those relevant situations are about missing objects, and the robot can easily find their alternatives in dining domains. By comparison, those situations involving "power", "water pipes", and "plug" are more difficult to the five methods (including COWP), because addressing those situations frequently requires skills beyond the robot's capabilities.

4.2.8 COWP versus baselines under LLMs

Table 5 provides a comparison of COWP's performance with three baselines: ProgPrompt, Inner Monologue, and LMZSP. The evaluation specifically concentrates on the success rate of task completion and the ability to handle various situations under different LLMs. These baselines leverage LLMs for planning, while our method is classical planning augmented by LLMs. We selected three LLM models for this comparison: text-ada-001, text-davinci-002, and text-davinci-003. Text-ada-001 is considered to be the least effective, while text-davinci-003 is viewed as the most powerful (OpenAI, 2023). From the table, it is clear that the choice of LLM has a significant impact on the baseline models' performance, with all models performing at their best under text-davinci-003. The baselines face challenges in completing task planning when utilizing the text-ada-001 model. The choice of LLM also impacts the performance of COWP, particularly when using text-ada-001. We observe that the performance of COWP under text-ada-001 becomes poor. Even when there is a situation where a robot is unable to complete a task, COWP continues with its current actions. For example, COWP will still use a dirty cup to serve water to humans. Comparing our COWP with its baselines, it is clear that the choice of LLM has a more substantial effect on the baseline models, as they heavily rely on the selected LLMs in their planning phase.

4.3 Robot demonstration

We demonstrated COWP using a mobile service robot that was tasked with delivering a cup for drinking water. The robot includes a UR5e arm, a Robotiq Hand-E gripper, and a Segway RMP-11 mobile base. The robot is capable of performing basic navigation and manipulation behaviors.

Table 5 The performances of COWP (ours) and three baseline methods (ProgPrompt, Inner Monologue, and LMZSP) are compared in terms of their task completion and situation handling percentages. The comparisons are made using different LLM models (text-davinci-003, text-davinci-002, and text-ada-001) for two service tasks

Task: wash glass	LLM models	COWP (ours)	ProgPrompt	Inner Monologue	LMZSP
Task completion percentage (%)	text-davinci-003	**59.3**	53.3	54.7	10.7
	text-davinci-002	**53.7**	36.0	27.3	5.3
	text-ada-001	**52.0**	0.0	0.0	0.0
Situation handling percentage (%)	text-davinci-003	**18.0**	14.0	12.0	0.0
	text-davinci-002	**14.0**	6.0	4.0	0.0
	text-ada-001	**6.0**	0.0	0.0	0.0
Task: serve water	LLM models	COWP (ours)	ProgPrompt	Inner Monologue	LMZSP
Task completion percentage (%)	text-davinci-003	**76.6**	69.3	32.7	6.7
	text-davinci-002	**73.3**	15.4	13.3	2.7
	text-ada-001	**66.0**	0.0	0.0	0.0
Situation handling percentage (%)	text-davinci-003	**34.7**	26.5	8.2	0.0
	text-davinci-002	**30.6**	8.2	6.1	0.0
	text-ada-001	**8.1**	0.0	0.0	0.0

Bold values represent the best results obtained in the respective experiments and comparisons

Specifically, we employed GG-CNN (Morrison et al., 2018) for object pick-and-place tasks. The navigation stack was built using the *move_base* package of the Robot Operating System (ROS) (Quigley et al., 2009). For visual scene analysis, the robot is equipped with a Robotiq Wrist Camera. With the assistance of Yolo-5 (Jocher et al., 2022), the robot recognizes objects in real time. Upon receiving a top-down image, Yolo-5 generates the coordinates of the bounding boxes and identifies the objects within them, thus providing an understanding of the geometric relationships between objects. We employ a heuristics-based algorithm to determine whether the target object is occupied by another, such as when bounding boxes overlap, or if the target object is absent. While there exist more powerful tools , such as vision-language models, for visual scene analysis, computer vision is not our focus, and our goal here is to implement a basic perception component to close the perceive-reason-act loop.

Figure 11 shows a real-world demonstration where a robot used COWP for planning to complete the service task of "Deliver a cup for drinking water." The initial plan included the following actions for the robot:

1. Walk to a cabinet on which a cup is located,
2. Grasp the cup after locating it,
3. Walk to dining table, and
4. Put down the cup to a table where the human is seated.

However, a situation was observed after executing Action #1, and the robot found *the cup is occupied*. The robot initially did not know whether this affects its plan execution. After querying GPT-3, the robot learned that one cannot pour water into an occupied cup, which renders its current plan infeasible. COWP enabled the robot to reason about other objects of the environment. The robot iteratively queried GPT-3 about whether X can be used for drinking water (using Prompt Template 2 in Sect. 3), where X is one of the objects from the environment. It happened that the robot learned "bowl" can be used for drinking water. With this newly learned, task-oriented commonsense knowledge, COWP successfully helped the robot generate a new plan, which used the bowl instead, to fulfill the service request.

We have generated a demo video that has been uploaded as part of the supplementary materials.

5 Conclusion and future work

In this paper, we develop a Large Language Model-based open-world task planning system for robots, called COWP, towards robust task planning and situation handling in open worlds. The novelty of COWP points to the integration of a classical, knowledge-based task planning system, and a pretrained language model for commonsense knowledge acquisition. The marriage of the two enables COWP to ground domain-independent commonsense knowledge to specific task planning problems. To evaluate COWP systematically, we collected a situation dataset that includes 1,085 execution-time situations in a dining domain. Experimental results suggest that COWP performed better than existing task planners developed for closed-world and open-world scenarios. We also provided a demonstration of COWP using a mobile manipulator working on delivery tasks, which provides a reference to COWP practitioners for real-world applications.

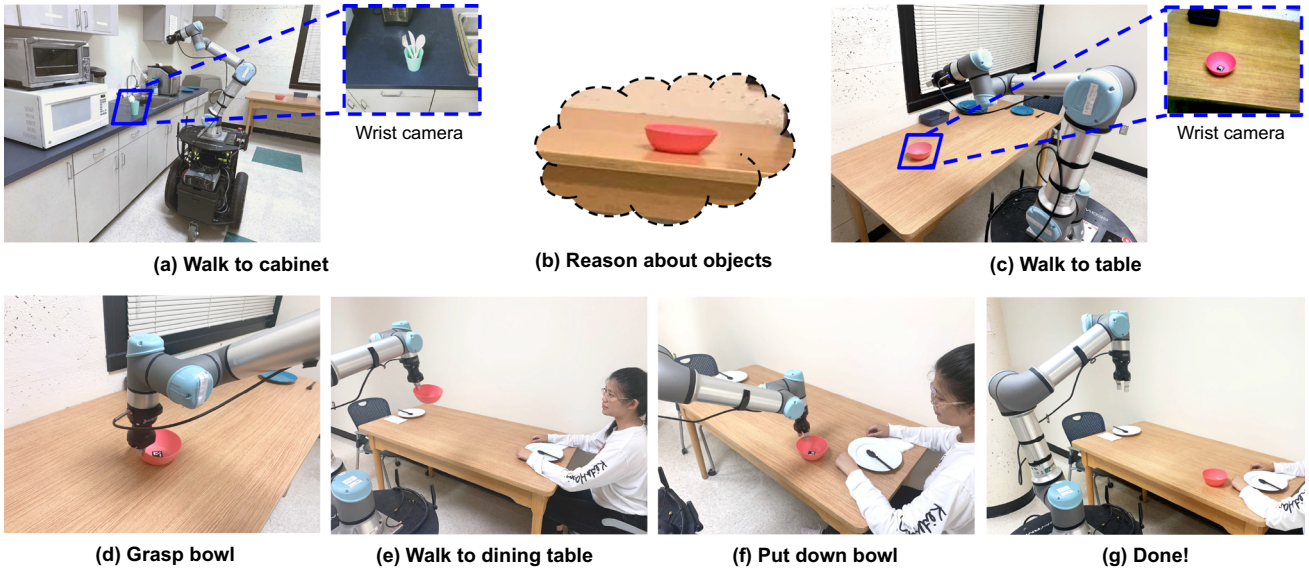

Fig. 11 An illustrative example of COWP for open-world planning, where the robot was tasked with "delivering a cup for drinking water." **a** The robot walked to a cabinet, and located a cup on the cabinet. However, the robot found a situation that there were objects in the cup (a knife, a fork, and a spoon in this case). This observation was entered into the plan monitor, which queried GPT-3, and suggested that the planned action "grasp" was not applicable given the occupied cup. Accordingly, COWP updated its task planner by adding the new information that one cannot pour water into a non-empty cup. **b** The robot reasoned about other objects that were available in the environment, and queried GPT-3 to update the task planner about whether those objects can be used for drinking water–details in Sect. 3. It happened that the robot learned a bowl could be used for drinking water. **c** A new plan of delivering a bowl to the human for drinking water was generated. Following the new plan, the robot walked to the table on which a bowl was located. **d** The robot grasped the bowl after observing it using vision. **e** The robot navigated to the dining table with the bowl. **f** The robot put down the bowl onto the dining table, and explained that a bowl was served due to the cup being occupied, which concluded the planning and execution processes. **g** The task is completed

We recognize the merits and limitations of "pure" LLM for planning (e.g., ProgPrompt) and classical planning augmented with LLMs, particularly in terms of logical consistency and flexibility of representation. Pure LLM planning has the advantage of greater flexibility, allowing it to handle a wider range of unforeseen situations, but it may be more prone to logical errors. In contrast, LLM-augmented classical planning, by parsing the LLM outputs into structured formats using hand-crafted rules, can reduce the occurrence of logical errors but may be less flexible when dealing with unforeseen situations. For instance, there might be situations where our hand-crafted rules fail to parse the output from LLMs.

It is evident that prompt design significantly affects the performance of Large Language Models (LLMs) (Zhang et al., 2021), which has motivated the recent research on prompt engineering. For instance, we tried replacing "suitable" with "possible" and "recommended," in the prompt templates and observed a decline in the system performance. Researchers can improve the prompt design of COWP in future work. There is the potential that other LLMs, such as ChatGPT (OpenAI, 2023) and Bard (Google, 2023), can produce better performance in open-world planning, and their performances might be domain-dependent, which can lead to very interesting future research. We present a complete implementation of COWP on a real robot, while acknowledging that there are many ways to improve the implementations. For instance, one can use a more advanced visual scene analysis tool to generate more informative observations for situation detection, or equip the robot with more skills (such as wiping a table, moving a chair, and opening a door) to deal with situations that cannot be handled now.

We acknowledge the limitations of our method in comprehensively understanding the world state in real-world systems. To effectively monitor plans, an autonomous system requires a general-purpose perception system capable of recognizing various situations, such as object states (dirty, broken, spilled, etc.) and event occurrences. While this task is challenging, recent developments in perception, such as vision-language models, hold promise in combining visual perception with language understanding to enhance autonomous systems' capability in detecting and interpreting situations (Zhang et al., 2023; Zhu et al., 2023; Gao et al., 2023). We are optimistic about overcoming these challenges in our future work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10514-023-10133-5.

Acknowledgements A portion of this work has taken place at the Autonomous Intelligent Robotics (AIR) Group, SUNY Binghamton. AIR research is supported in part by grants from the National Science Foundation (NRI-1925044), Ford Motor Company, OPPO, and SUNY Research Foundation.

Author Contributions YD, XZ, SA, HY, AK, CE, and SZ contributed to the development of the initial ideas and methodology. YD, XZ, and SA contributed to implementing the methodology. YD, XZ, SA, and NC contributed to the experiments. YD, XZ, SA, HY, and SZ contributed to the analysis of the results. YD, XZ, SA, and SZ contributed to the manuscript writing. All authors reviewed and provided feedback on the manuscript.

Funding A portion of this work has taken place at the Autonomous Intelligent Robotics (AIR) Group, SUNY Binghamton. AIR research is supported in part by grants from the National Science Foundation (NRI-1925044), Ford Motor Company, OPPO, and SUNY Research Foundation

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., et al. (1998). *PDDL| the planning domain definition language*. Tech Rep: Technical Report.

Amiri, S., Bajracharya, S., Goktolgal, C., Thomason, J., & Zhang, S. (2019). Augmenting knowledge through statistical, goal-oriented human-robot dialog. In: *2019 IEEE/RSJ International conference on intelligent robots and systems (IROS)*. IEEE; 2019. p. 744–750.

Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., & Ho, D., et al. (2023a). Do as i can, not as i say: Grounding language in robotic affordances. In: *Conference on robot learning*; 287–318.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al. (2023b). RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control, arXiv preprint arXiv:2307.15818.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems., 33*, 1877–1901.

Chen, M., Tworek, J., Jun, H., & Yuan, Q. (2021). Pinto HPdO, Kaplan J, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Chernova, S., Chu, V., Daruna, A., Garrison, H., Hahn, M., & Khante, P. et al. (2020) Situated bayesian reasoning framework for robots operating in diverse everyday environments. In: *Robotics research*. Springer; . p. 353–369.

Davis, E., & Marcus, G. (2015). Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM, 58*(9), 92–103.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019) Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of naacL-HLT*, 1(2)-2.

Ding, Y., Zhang, X., Paxton, C., & Zhang, S. (2023). Task and Motion Planning with Large Language Models for Object Rearrangement. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Elsweiler, D., Hauptmann, H., & Trattner, C. (2022). Food recommender systems. In: *Recommender systems handbook*. Springer; 871–925.

Galindo, C., Fernández-Madrigal, J. A., González, J., & Saffiotti, A. (2008). Robot task planning using semantic maps. *Robotics and Autonomous Systems, 56*(11), 955–966.

Gao, P., Han, J., Zhang, R., Lin, Z., Geng, S.,& Zhou, A.,et al. (2023). Llama-adapter v2: Parameter-efficient visual instruction model. arXiv preprint arXiv:2304.15010.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2020). Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. *Proceedings of the international conference on automated planning and scheduling., 30*, 440–448.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., et al. (2021). Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems, 4*, 265–293.

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge University Press.

Google.: Bard FAQ. Accessed on April 7, (2023). https://bard.google.com/faq.

Hanheide, M., Göbelbecker, M., Horn, G. S., Pronobis, A., Sjöö, K., Aydemir, A., et al. (2017). Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence, 247*, 119–150.

Haslum, P., Lipovetzky, N., Magazzeni, D., & Muise, C. (2019). An introduction to the planning domain definition language. *Synthesis Lectures on Artificial Intelligence and Machine Learning, 13*(2), 1–187.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research, 26*, 191–246.

Hoffmann, J. (2001). FF: The fast-forward planning system. *AI magazine, 22*(3), 57–57.

Huang, W., Abbeel, P., Pathak, D., Mordatch, I. (2022) Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *Thirty-ninth international conference on machine learning*.

Huang, W., Xia, F., Shah, D., Driess, D., Zeng, A., & Lu, Y., et al.(2023). Grounded Decoding: Guiding text generation with grounded models for robot control. arXiv preprint arXiv:2303.00855.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., & Florence, P., et al. (2022). Inner Monologue: Embodied Reasoning through Planning with Language Models. In: *6th Annual conference on robot learning*.

Jiang, Y., Walker, N., Hart, J., &Stone, P. (2019) Open-world reasoning for service robots. In: *Proceedings of the international conference on automated planning and scheduling*. vol. 29; . p. 725–733.

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., Kwon, Y., & Michael, K., et al. (2022). ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation. Zenodo.

Kant, Y., Ramachandran, A., Yenamandra, S., Gilitschenski, I., Batra, D., & Szot, A. et al. (2022) Housekeep: Tidying virtual households using commonsense reasoning. In: *Computer vision–ECCV* 2022. Springer; . p. 355–373.

Knoblock C.A.,& Tenenberg, J.D., Yang, Q. (1991) Characterizing abstraction hierarchies for planning. In: *Proceedings of the ninth national conference on artificial intelligence*2692–697.

Li, S., Puig, X., Paxton, C., Du, Y., Wang, C., & Fan, L., et al. (2022). Pre-trained language models for interactive decision-making. Advances in Neural Information Processing Systems.

Lin, K., Agia, C., Migimatsu, T., Pavone, M., & Bohg, J. (2023). Text2Motion: From natural language instructions to feasible plans. arXiv preprint arXiv:2303.12153.

Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., & Biswas, J., et al. (2023). LLM+P: Empowering large language models with optimal planning proficiency. arXiv preprint arXiv:2304.11477.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys, 55*(9), 1–35.

Lo, S. Y., Zhang, S., & Stone, P. (2020). The petlon algorithm to plan efficiently for task-level-optimal navigation. *Journal of Artificial Intelligence Research., 69*, 471–500.

Morrison, D., Corke, P., & Leitner, J. (2018). Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. In: *Robotics: Science and Systems (RSS)*.

Nau, D. S., Au, T. C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., et al. (2003). SHOP2: An HTN planning system. *Journal of artificial intelligence research, 20*, 379–404.

OpenAI.: ChatGPT. Cit. on pp. 1, 16. Accessed: 2023-02-08. Retrieved from: https://openai.com/blog/chatgpt/.

OpenAI.: GPT-4 technical report.

OpenAI.: Models–OpenAI API. Retrieved: 2023-07-10. https://platform.openai.com/docs/models/overview.

Perera, V., Soetens, R., Kollar, T., Samadi, M., Sun, Y., Nardi, D., et al. (2015). Learning task knowledge from dialog and web access. *Robotics, 4*(2), 223–252.

Puig, X., Ra, K., Boben, M., Li, J., Wang, T., & Fidler, S., et al. (2018). Virtualhome: Simulating household activities via programs. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*; 2018. 8494–8502.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., & Leibs, J., et al. (2009). ROS: an open-source Robot Operating System. In: *ICRA workshop on open source software*. vol. 3. Kobe, Japan; p. 5.

Reiter, R. (1981) On closed world data bases. In: *Readings in artificial intelligence*. Elsevier. p. 119–140.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., & Tremblay, J., et al. (2023). Progprompt: Generating situated robot task plans using large language models. *International Conference on Robotics and Automation (ICRA)*.

Song, CH., Wu, J., Washington, C., Sadler, BM., Chao, WL., & Su, Y. (2023). Llm-planner: Few-shot grounded planning for embodied agents with large language models. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Tucker, M., Aksaray, D., Paul, R., Stein, G.J., & Roy, N.(2020) Learning unknown groundings for natural language interaction with mobile robots. In: *Robotics research*. Springer; 317–333.

Valmeekam, K., Olmo, A., Sreedharan, S., & Kambhampati, S. (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In: *Foundation Models for Decision Making Workshop at Neural Information Processing Systems*.

Valmeekam, K., Sreedharan, S., Marquez, M., Olmo, A., & Kambhampati, S. (2023). On the planning abilities of large language models (a critical investigation with a proposed benchmark). arXiv preprint arXiv:2302.06706. 2023;.

Wang, C., Liu, P., & Zhang, Y. (2021). Can generative pre-trained language models serve as knowledge bases for closed-book QA? In: *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing*; 3241–3251.

West, P., Bhagavatula, C., Hessel, J., Hwang, JD., Jiang, L., & Bras, RL, et al. (2022). Symbolic knowledge distillation: From general language models to commonsense models. *Proceedings of the 2022 conference of the North American chapter of the association for computational linguistics: Human language technologies*.

Xie, Y., Yu, C., Zhu, T., Bai, J., Gong, Z., & Soh, H.(2023). Translating natural language to planning goals with large-language models. arXiv preprint arXiv:2302.05128.

Yq, Jiang, Sq, Zhang, Khandelwal, P., & Stone, P. (2019). Task planning in robotics: An empirical comparison of PDDL-and ASP-based systems. *Frontiers of Information Technology & Electronic Engineering., 20*(3), 363–373.

Zhao, Z., Lee, WS., & Hsu, D (2023). Large Language Models as Commonsense Knowledge for Large-Scale Task Planning, *RSS Workshop on Learning for Task and Motion Planning*

Zhang, X., Ding, Y., Amiri, S., Yang, H., Kaminski, A.,& Esselink, C, et al. (2023). Grounding classical task planners via vision-language models. In: *ICRA Workshop on Robot Execution Failures and Failure Management Strategies*.

Zhang, N., Li, L., Chen, X., Deng, S., Bi, Z., & Tan, C. et al (2021). ifferentiable prompt makes pre-trained language models better few-shot learners. In: *International conference on learning representations*.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., & Chen, S., et al. (2022). OPT: Open pre-trained transformer language models. arXiv preprint arXiv:2205.01068.

Zhu, D., Chen, J., Shen, X., Li, X., & Elhoseiny, M. (2023). Minigpt-4: Enhancing vision-language understanding with advanced large language models. arXiv preprint arXiv:2304.10592.

Yan Ding is a Ph.D. candidate in Computer Science at Binghamton University, USA. He holds a master's and a bachelor's degree from Chongqing University, China. His research focuses on the intersection of planning and learning in complex domestic environments, employing methodologies from classical planning, task and motion planning, machine learning, and reinforcement learning.

Xiaohan Zhang is a Ph.D. student in the department of computer science at The State University of New York at Binghamton, starting from Fall 2020. Xiaohan received a B.S. degree at Renmin University of China in 2019. His research goal is to develop algorithms that solve complicated and long-horizon tasks for autonomous robots. His research touches robot planning (e.g., probabilistic planning, task and motion planning) and learning (e.g., classical machine learning, deep learning, reinforcement learning).

Saeid Amiri received his Ph.D. degree from SUNY Binghamton (2022). He received his M.Sc. and B.Sc. in Mechanical Engineering at University of Houston (2015) and Sharif University of Technology (2013) where his primary focus was on classical control. During Ph.D., his main research has been on sequential decision making algorithms under partial observability that leverage human knowledge and past experiences. He has applied these algorithms to robotic spoken dialogue systems, human-robot interaction, and target search domains. In 2020, he interned at ABB Robotics. He is interested in using mobile and manipulator robotic platforms.

Nieqing Cao is currently pursuing a Ph.D. degree at the Department of Systems Science and Industrial Engineering, the State University of New York at Binghamton, Binghamton, NY, USA. Her research interests include large-scale data predictive modeling and systems optimization in manufacturing.

Hao Yang received a Ph.D. degree from the University of Missouri-Rolla, where he conducted research on the application of case-based reasoning in manufacturing. He had been employed at Ford Motor Company for over 28 years. During his tenure at Ford, he initiated and contributed to a multitude of AI-related projects and assumed leadership roles in various engineering software teams, overseeing the development of diverse software projects. His AI interests encompass case-based reasoning, planning, machine learning, natural language processing, and dialogue systems. Lately, his curiosity has gravitated towards autonomous robots interacting with humans in open-world scenarios and topics related to cognitive science.

Andy Kaminski With a 25-year tenure engaged with software-oriented endeavors at Ford Motor Company, Andy has showcased his mastery across diverse domains of software development. From pioneering cloud architecture solutions, advancing edge computing architectures, to harnessing the power of AI.

Chad Esselink is a former Senior Manager of Edge Software Research, working with software at Ford Motor Company for 29 years. Chad received a M.S. Degree from Rensselaer Polytechnic Institute in 2004 and a B.S. Degree from Michigan Technological University in 1994. Chad is an inventor on 19 Vehicle Connectivity patents and 1 Manufacturing Quality patent. His research goals relate to exploring edge software integrating cloud and embedded systems.

Shiqi Zhang is an Associate Professor of Computer Science, The State University of New York at Binghamton (SUNY Binghamton). From 2014 to 2016, he was a Postdoctoral Fellow working on a team of mobile service robots at UT Austin. He worked at Cleveland State University from 2016-2018. He received his Ph.D. in Computer Science (2013) from Texas Tech University, and received his M.S. (2008) and B.S. (2006) degrees from Harbin Institute of Technology. Dr. Zhang's research lies in the intersection of artificial intelligence and robotics. He works on developing intelligent mobile robots that are able to interact with people, provide services to people, and learn from this experience, in human-robot collaborative environments.