A Questionnaire for Collecting Situation Dataset

To collect execution-time situations, a questionnaire was designed and published on the Amazon Mechanical Turk. Figure 6 shows the Mechanical Turk interface for one everyday task (i.e., drinking water). In the interface, 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 within the blank. On the questionnaire, there are 12 everyday tasks (e.g., setting a dining table) associated with their steps, which were extracted from an existing dataset [11]. In the end, we have collected a dataset of 1128 valid situations, where each instance of the dataset corresponds to a situation that prevents a service robot from completing a task in a dining domain. In the next section, we will discuss the statistics of the dataset.

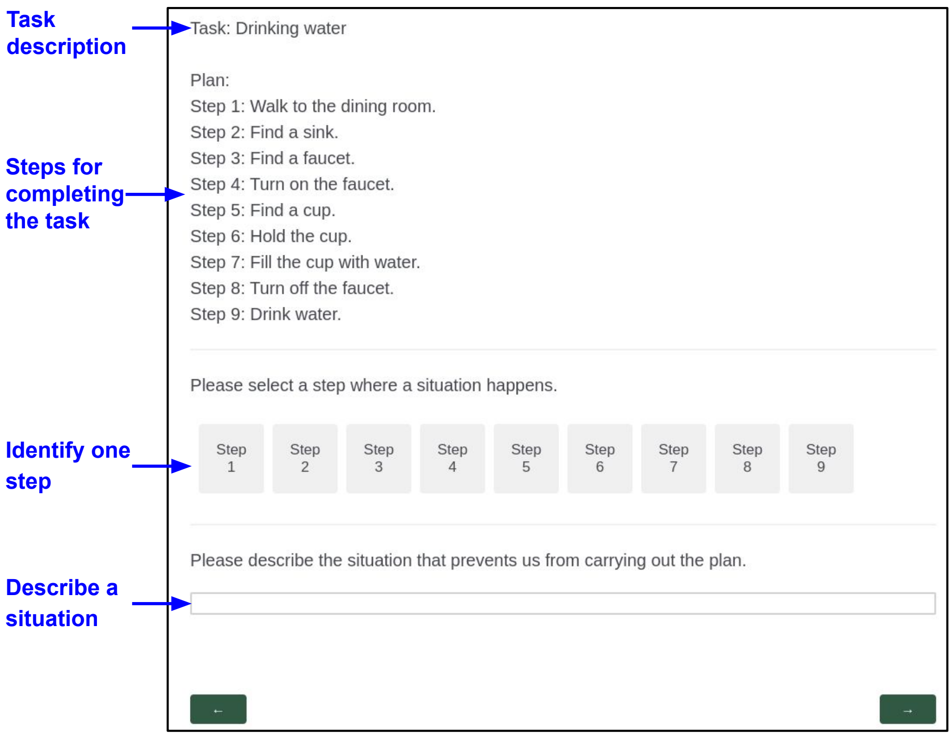


Figure 6: The Mechanical Turk interface for the task of drinking water. Each MTurker was provided with a task description, including nine steps for completing the task. The MTurkers were asked to respond to a questionnaire by identifying one step (from Steps 1 to 9) in the provided plan, and describing a situation that might occur in that step.

B Statistics of Situation Dataset

Figures 7 and 8 show the statistics of situations for six everyday tasks used in our evaluation, where *x-axis* reflects the occurrence of each *distinguishable situations*, and *y-axis* represents each distinguishable situations, respectively. In the top left corner of each subfigure, (X) represents the number of distinguishable situations in each task. In the bottom right corner of each subfigure, Total = X represents the number of *situations* in each task. According to the two figures, we can see that there are at least 92 situations collected for each of the six tasks used in our evaluation, with 16 to 22 distinguishable situations.

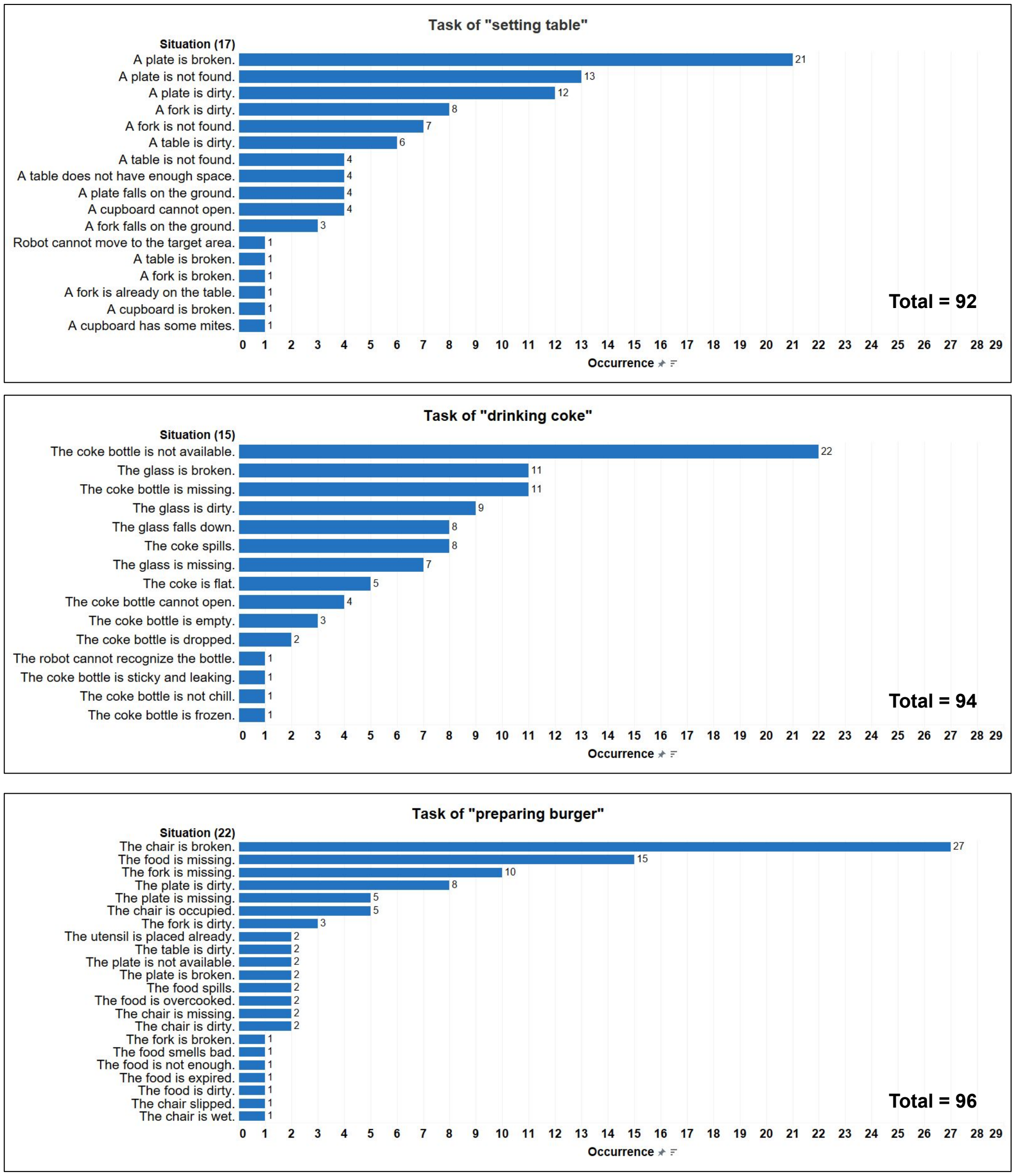


Figure 7: **Top:** Details of situations in the task of "setting table"; **Middle:** Details of situations in the task of "drinking coke"; **Bottom:** Details of situations in the task of "preparing burger"; *x-axis* reflects the occurrence of each *distinguishable situations*, and *y-axis* represents each distinguishable situations, respectively. (X) in the top left corner of each subfigure represents the number of distinguishable situations in each task. Total = X indicates the number of situations in each task.

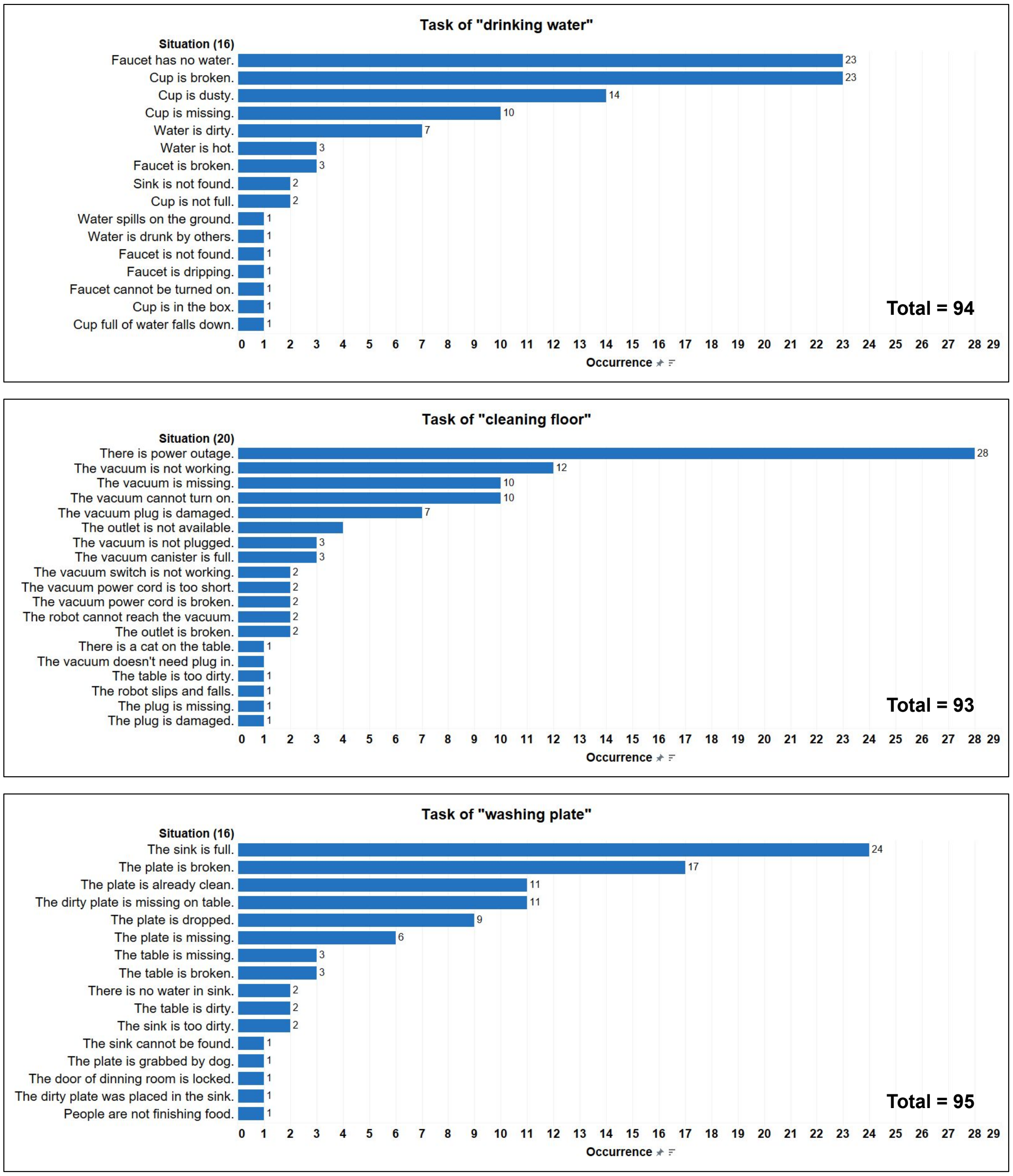


Figure 8: **Top:** Details of situations in the task of "drinking water"; **Middle:** Details of situations in the task of "cleaning floor"; **Bottom:** Details of situations in the task of "washing plate"; *x-axis* reflects the occurrence of each *distinguishable situations*, and *y-axis* represents each distinguishable situations, respectively. (X) in the top left corner of each subfigure represents the number of distinguishable situations in each task. Total = X indicates the number of situations in each task.

C Object Library for Simulation

For simulating dining tasks, we extracted 86 objects (e.g., cup, burger, folk, table, and chair) from an existing dataset [11]. Figure 9 shows these objects, which is categorized into five groups: utensil, appliance, furniture, food, and beverage. From the figure, we can see that the category "utensil" contains the greatest number of objects (i.e., 29), while the category "beverage" contains the fewest ones (i.e., 8).

Utensil (29)	Appliance (18)	Furniture (16)	Food (15)	Berevage (8)
dish	blender	bookshelf	bread	beer
cleaning bottle	dishwasher	closet	cake	milk
cooking pot	freezer	cpu table	ice cream	watermelon juice
frying pan	microwave	cupboard	noodles	alcohol
trash can	fridge	desk	oatmeal	coffee
cloth napkin	oven	kitchen cabinet	peanut butter	juice
paper towel	stove	nightstand	rice	tea
bucket	washing machine	wooden chair	salt	wine
coffee cup	kettle	piano bench	snack	
colander	vacuum cleaner	table cloth	sugar	
condiment bottle	toaster	coffee table	oil	
dish bowl	air fryer	couch	pasta	
drinking glass	dehumidifier	dining table	chips	
measuring cup	water boiler	kitchen table	sauce	
mug	ice cream maker	kitchen counter	steak	
rag	juicer	pantry		
wine glass	water filter			
chef knife	coffee maker			
condiment shaker				
cutlery fork				
cutlery knife				
cutting board				
dish rack				
oven tray				
mat				
wooden spoon				
sponge				
coffee filter				
wooden chopstick				

Figure 9: For simulating dining tasks, we extracted 86 objects from an existing dataset [11]. These objects are categorized into five groups: utensil, appliance, furniture, food, and beverage, with (X) representing the number of objects in each group.

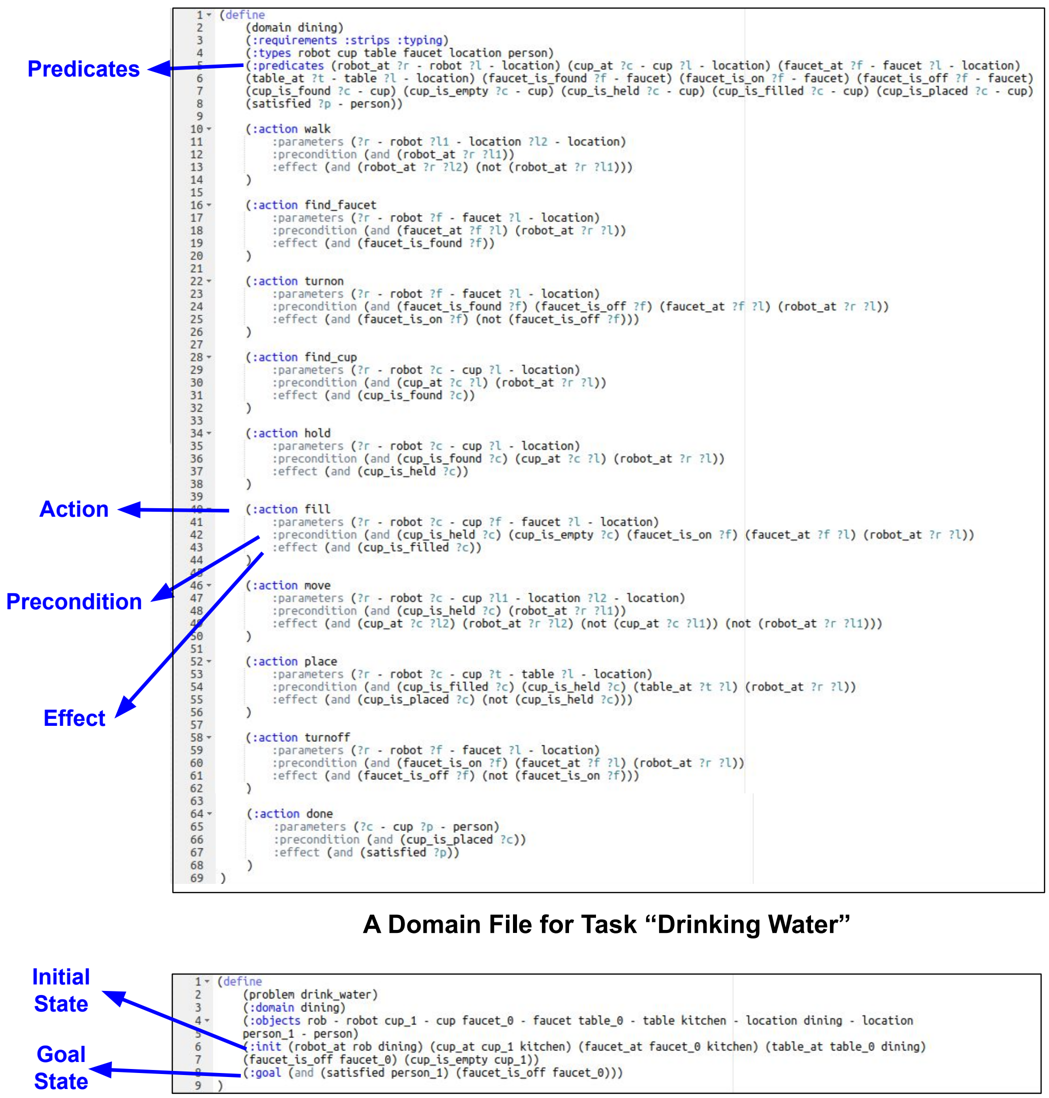
D Closed-World Task Planners in PDDL

For each task in the evaluation, we developed a closed-world task planner in PDDL. PDDL, an action-centred language, is designed to formalize Artificial Intelligence (AI) planning problems, allowing for a more direct comparison of planning algorithms and implementations [35]. Figure 10 shows a task planner for the task of "drinking water", which consists of a domain file (**upper**) and a problem file (**lower**). In the upper subfigure, a set of predicates (e.g., `cup_at`) and a set of actions (e.g., `fill`) are predefined, where an action is defined by its preconditions and effects. For example, one of preconditions for action `fill` is `(cup_is_held ?c) ∧ (cup_is_empty ?c)`, and the action effect is `(cup_is_filled ?c)`. In the lower subfigure, a task problem is defined by an initial state and a goal state (i.e., a user is satisfied and the faucet is turned off.) A task plan for drinking water is generated after inputting these two files into a solver[2], as shown below:

[2]The solver is accessible at http://editor.planning.domains/

```
______ Task Plan for Drinking Water ______

S1:  (walk rob dining kitchen)
S2:  (find_faucet rob faucet_0 kitchen)
S3:  (find_cup rob cup_1 kitchen)
S4:  (hold rob cup_1 kitchen)
S5:  (turnon rob faucet_0 kitchen)
S6:  (fill rob cup_1 faucet_0 kitchen)
S7:  (turnoff rob faucet_0 kitchen)
S8:  (walk rob kitchen dining)
S9:  (place rob cup_1 table_0 dining)
S10:  (done cup_1 person_1)
```




```
(define
    (domain dining)
    (:requirements :strips :typing)
    (:types robot cup table faucet location person)
    (:predicates (robot_at ?r - robot ?l - location) (cup_at ?c - cup ?l - location) (faucet_at ?f - faucet ?l - location)
    (table_at ?t - table ?l - location) (faucet_is_found ?f - faucet) (faucet_is_on ?f - faucet) (faucet_is_off ?f - faucet)
    (cup_is_found ?c - cup) (cup_is_empty ?c - cup) (cup_is_held ?c - cup) (cup_is_filled ?c - cup) (cup_is_placed ?c - cup)
    (satisfied ?p - person))

    (:action walk
        :parameters (?r - robot ?l1 - location ?l2 - location)
        :precondition (and (robot_at ?r ?l1))
        :effect (and (robot_at ?r ?l2) (not (robot_at ?r ?l1)))
    )

    (:action find_faucet
        :parameters (?r - robot ?f - faucet ?l - location)
        :precondition (and (faucet_at ?f ?l) (robot_at ?r ?l))
        :effect (and (faucet_is_found ?f))
    )

    (:action turnon
        :parameters (?r - robot ?f - faucet ?l - location)
        :precondition (and (faucet_is_found ?f) (faucet_is_off ?f) (faucet_at ?f ?l) (robot_at ?r ?l))
        :effect (and (faucet_is_on ?f) (not (faucet_is_off ?f)))
    )

    (:action find_cup
        :parameters (?r - robot ?c - cup ?l - location)
        :precondition (and (cup_at ?c ?l) (robot_at ?r ?l))
        :effect (and (cup_is_found ?c))
    )

    (:action hold
        :parameters (?r - robot ?c - cup ?l - location)
        :precondition (and (cup_is_found ?c) (cup_at ?c ?l) (robot_at ?r ?l))
        :effect (and (cup_is_held ?c))
    )

    (:action fill
        :parameters (?r - robot ?c - cup ?f - faucet ?l - location)
        :precondition (and (cup_is_held ?c) (cup_is_empty ?c) (faucet_is_on ?f) (faucet_at ?f ?l) (robot_at ?r ?l))
        :effect (and (cup_is_filled ?c))
    )

    (:action move
        :parameters (?r - robot ?c - cup ?l1 - location ?l2 - location)
        :precondition (and (cup_is_held ?c) (robot_at ?r ?l1))
        :effect (and (cup_at ?c ?l2) (robot_at ?r ?l2) (not (cup_at ?c ?l1)) (not (robot_at ?r ?l1)))
    )

    (:action place
        :parameters (?r - robot ?c - cup ?t - table ?l - location)
        :precondition (and (cup_is_filled ?c) (cup_is_held ?c) (table_at ?t ?l) (robot_at ?r ?l))
        :effect (and (cup_is_placed ?c) (not (cup_is_held ?c)))
    )

    (:action turnoff
        :parameters (?r - robot ?f - faucet ?l - location)
        :precondition (and (faucet_is_on ?f) (faucet_at ?f ?l) (robot_at ?r ?l))
        :effect (and (faucet_is_off ?f) (not (faucet_is_on ?f)))
    )

    (:action done
        :parameters (?c - cup ?p - person)
        :precondition (and (cup_is_placed ?c))
        :effect (and (satisfied ?p))
    )
)
```

A Domain File for Task "Drinking Water"

```
(define
    (problem drink_water)
    (:domain dining)
    (:objects rob - robot cup_1 - cup faucet_0 - faucet table_0 - table kitchen - location dining - location
    person_1 - person)
    (:init (robot_at rob dining) (cup_at cup_1 kitchen) (faucet_at faucet_0 kitchen) (table_at table_0 dining)
    (faucet_is_off faucet_0) (cup_is_empty cup_1))
    (:goal (and (satisfied person_1) (faucet_is_off faucet_0)))
)
```

A Problem File for Task "Drinking Water"

Figure 10: A closed-world task planner in PDDL for the task of "drinking water", consisting of a domain file (**upper**) and a problem file (**lower**). In the upper subfigure, a set of predicates and a set of actions are predefined, where an action is defined by its preconditions and effects. In the lower subfigure, a task problem is defined by an initial state and a goal state.

E Prompt Design

Figure 11 shows three examples for prompt construction based on our Templates 1-3, respectively. In the figure, the interface, called Playground[3], is intended for testing GPT-3 online, where a user can text a prompt in the blank, and customize the hyperparameters of GPT-3 (e.g., model). In our case, we use the *text-davinci-002* model, which is the most capable engine. The prompt for Plan Monitor (PM) is constructed based on Template 1, where PM evaluates if the current task plan is feasible or not. In the **top** figure, we can know an action precondition that "one cannot fill a broken cup with water" according to the common sense from GPT-3. The prompts for Knowledge Acquirer (KA) are constructed based on Template 2 and Template 3, where KA extracts common sense to augment the classical task planner. In these two figures (**middle** and **bottom**), we can know that common sense that "one can use a bowl for drinking water" can be added into an action effect, according to the common sense from GPT-3.

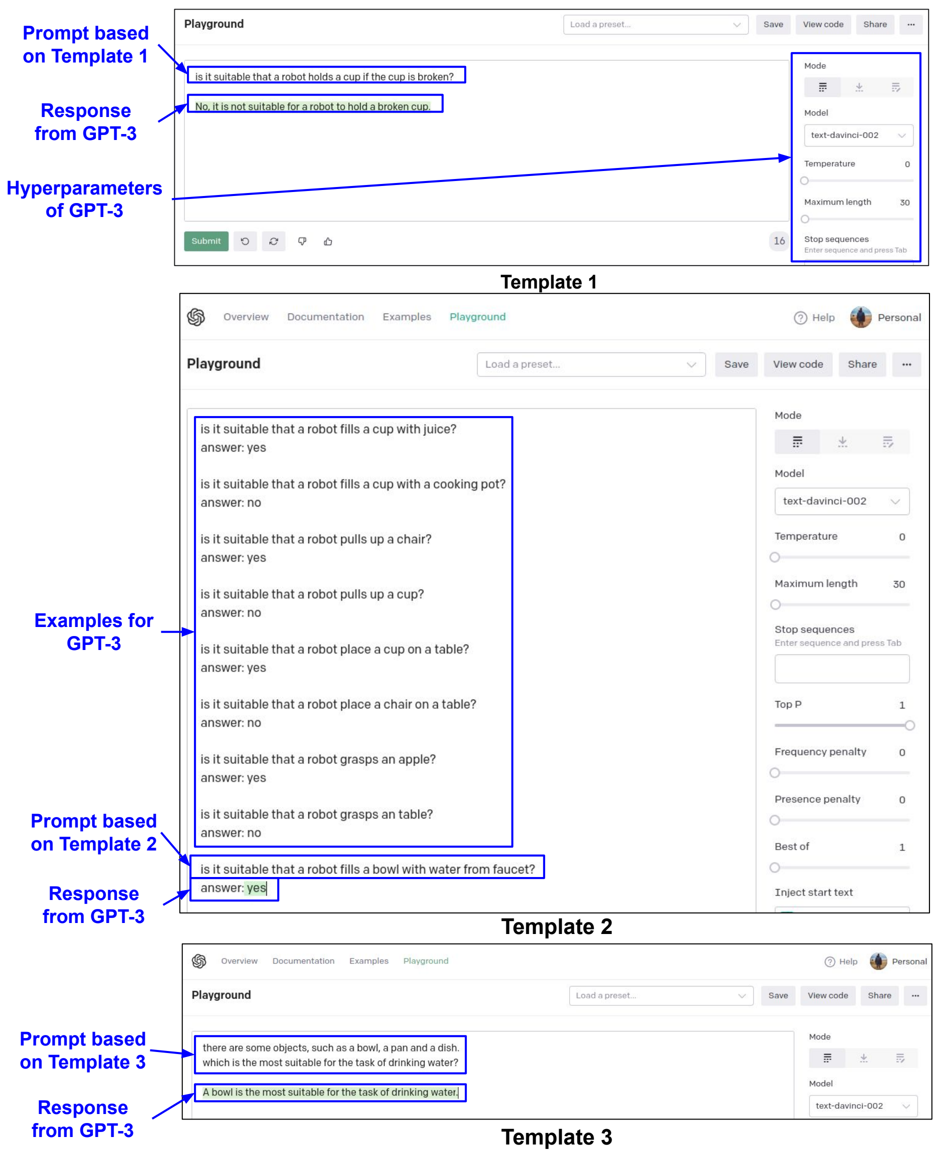


Figure 11: Three examples for prompt construction based on Templates 1-3.

[3]The playground of GPT-3 is accessible at https://beta.openai.com/playground.

F Implementation of ThreeDWorld

ThreeDWorld, a platform for physical simulation, is applied for visualization. The implementation consists of three components, as shown in Figure 12. First, the scene of a dining domain is generated, including different objects such as a table, a microwave, and a sink. Second, a situation is randomly generated based on our situation dataset. Third, our robot executes the robust task plan generated by COWP. In our case, we use the Magnebot[4], a virtual robot provided by ThreeDWorld, which is capable of performing high-level actions, such as moving to an object and picking up an object with "magnets."

We have generated a **demo video** that can be accessed in the supplementary material. In this video, the robot finds a situation that "the mug is broken" in the task of "drinking water", and a bowl is eventually grasped towards task completion and situation handling.

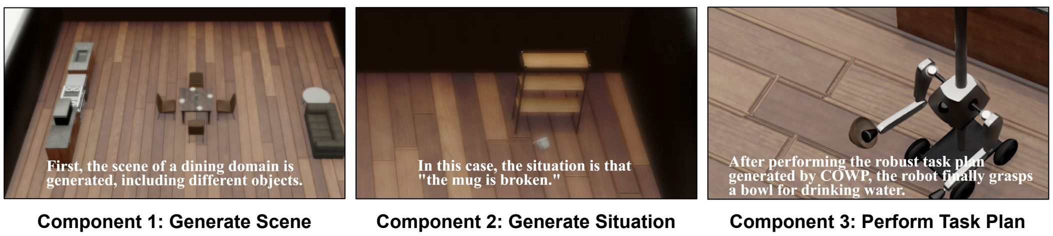


Figure 12: ThreeDWorld, a platform for physical simulation, is used for visualization, where the implementation consists of three cantonments, i.e., generating the scene of a dining domain, generating a situation, and performing the robust task plan generated by COWP.

[4]An introduction to the Magnebot in ThreeDWorld: https://github.com/alters-mit/magnebot