SlugJARVIS: Multimodal Commonsense Knowledge-based Embodied AI for SimBot Challenge

Jing Gu* **Kaizhi Zheng*** **Kaiwen Zhou** **Yue Fan**

Xuehai He **Jialu Wang** **Zonglin Di** **Xin Eric Wang**
University of California, Santa Cruz
{jgu110, kzheng31, kzhou35, yfan71, xhe89, jwang470, zdi, xwang366}@ucsc.edu

Abstract

We present SlugJARVIS, an advanced multimodal, task-oriented embodied AI developed for the purpose of assisting users in navigating complex real-world tasks within the context of the Alexa Prize SimBot Challenge. SlugJARVIS is designed to effectively maintain user engagement, accurately interpret user intent, make informed and executable decisions based on the current environment, and adeptly manage unforeseen situations. This functionality is achieved through a comprehensive system that encompasses single- and multimodal understanding, systematic task management, and goal-oriented response generation.

1 Introduction

The goal of embodied AI in robot task completion is to create robots that are more versatile and adaptable to a range of different environments and tasks for daily activities. An embodied agent is expected to have the following capabilities: (1) communicate with humans and understand their intents, (2) perceive the environment and recognize objects using visual information, and (3) plan and execute sequences of actions to achieve the desired goals. Recently, Alexa Arena [3], a user-centric simulation platform, was presented to facilitate research in building autonomous embodied robots. In each Arena game, the user will be prompted with the high-level goals of the task, such as "change the color of a white bowl to red," while the robot agent is blind to such mission goals. The users can engage in real-time dialogue with robot agents to assist them in completing the tasks. The agent will be rated according to the progress of the task completion and the participant experiences.

We present SlugJARVIS, a versatile AI for household tasks and human interaction, consisting of modality understanding (section 2), knowledge (section 3), task execution (section 4), and response generation modules (section 5). Our bot can communicate in natural language, navigate, and interact with the environment to complete tasks on the Alexa Arena simulation platform [3]. SlugJARVIS processes complex information from the environment and users. Upon receiving user input, alongside the current user interface image and previous action and environment status, the bot generates formulated subgoals. The knowledge module refines these subgoals with stored information, producing executable actions in the Arena environment. The response module then creates natural language responses converted to speech based on planned actions and user input. Finally, SlugJARVIS awaits the user's next input. There are five aspects that make SlugJARVIS more favorable:

Large Language Model Reasoning We emphasize the application of large language models in the planning process, particularly in addressing corner cases and handling ASR failures. We generate subgoals from input language instructions while efficiently managing GPU memory constraints. To tackle ASR failures, we have experimented with various techniques that incorporate the principles

*Equal Contribution

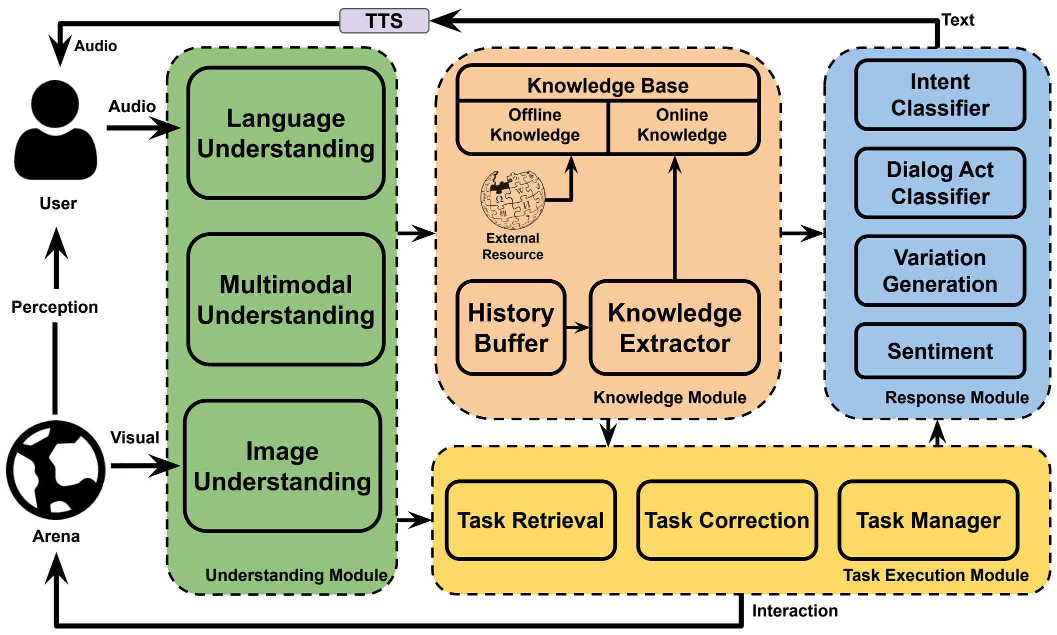


Figure 1: **System overview.**

of curriculum learning and continual learning, showcasing the versatility and adaptability of large language models in our approach.

Reasoning with Multimodal Information Our agent is capable of aggregating visual information observed in the environment and textual information from user utterances. In particular, we leverage commonsense knowledge to associate the entities parsed in human language with the recognized objects. This specialized neuro-symbolic reasoning design can help our agent better generalize to unseen environments.

Flow-guided Task Management SlugJARVIS is an intelligent embodied agent that processes multimodal information from vision, text, and action as input and navigates different game environments. Considering the complexity of the game, we designed a control system that manages the task flow. First, SlugJARVIS utilizes a progressive task experience retriever to identify the current playing task. The progressive task experience retrieval algorithm enhances SlugJARVIS's ability to adapt to various environments and tasks by leveraging past successful experiences. In addition, we designed a Task Manager comprising a subgoal manager and an action manager. During the inference phase, the subgoal manager integrates external knowledge with subgoals to generate an action list. The action manager translates actions from the action list into executable actions.

Acquiring Knowledge Base through Online Learning Robots greatly benefit from environmental knowledge when it comes to accomplishing tasks. Actively acquiring knowledge about a new environment equips the bot with the ability to swiftly locate targets and prevent getting stuck during gameplay. Our bot's knowledge base is designed to continuously update with each new interaction, ensuring adaptability and improved performance.

SimBot-oriented Chatbot Alexa users employ natural language to communicate with our SlugJARVIS, and a sophisticated chatbot is essential for smooth task progress and a great user experience. First, to avoid off-topic conversation, we designed an end-to-end topic detection model to monitor and control the dialogue flow. We also utilized a profanity module to avoid toxic conversations. We built a commonsense-based parsing system to accurately estimate users' intentions. SlugJARVIS will generate a response considering action execution results, user intentions, and task progress. Additionally, to improve user experience, SlugJARVIS contains several auxiliary dialogue modules such as information-prompt, where our bot would prompt users how to play the game if a user decision dilemma is detected. In this technical report, we will systematically introduce the underlying design of SlugJARVIS and present qualitative analysis to demonstrate its effectiveness.

2 Understanding Module

An embodied agent in SimBot needs to consider information from various modalities including vision, text, depth, action, and action execution results. Here we introduce our understanding module to process the complicated information.

2.1 Language Understanding

Utilizing natural language, users convey their intentions directly; therefore, comprehending the current user language is of paramount importance.

Coreference. Initially, coreference resolution is conducted. To elaborate, an end-to-end coreference module from the AllenNLP package [5] is employed to detect and transform all pronouns into corresponding nouns mentioned in the dialogue history, followed by the addition of supplementary resolution rules for post-processing purposes, such as addressing elongated nouns.

Off-topic Handling. Given the possibility of Alexa Echo users engaging in irrelevant discussions, an end-to-end model was constructed to identify such occurrences. Off-topic models were trained as a classification task to discern if a user response pertained to the Alexa Prize SimBot challenge. In-topic data comprised all utterances from the chatbot's dialogue history, while off-topic data consisted of random utterances from online chats. The dataset, containing 331,234 in-topic and 892,223 off-topic utterances, was divided into train, development, and test sets in an 80/10/10 ratio. A BERT-base model [2] served as the foundation, with an additional linear layer for classification purposes. The model achieved 99.92% accuracy on the test set and demonstrated proficiency in identifying off-topic content during online interactions. A profanity model was also implemented for toxic language detection.

Rule-based Subgoal Parser. To streamline dialogue flow management, an intricate rule-based subgoal parser was developed to transform each utterance into a subgoal. These subgoals provide a clear representation of user intent. For instance, the utterance "Could you go to the breakroom and open the printer?" would be converted into "1. ['Goto, Breakroom']; 2. ['Turn on, Printer']." Additionally, the system incorporates supplementary commonsense knowledge. For example, if the chatbot received the utterance "I need you to find the apple" but failed to locate it in the current room, further subgoals such as "['Open, Refrigerator']" would be generated to inspect the refrigerator, as the commonsense knowledge corpus suggests that apples might be stored there.

Large Language Model. In addition to rule-based methods for subgoal generation, we also employ large language models, specifically the PyThia model [1], for handling certain scenarios, particularly corner cases. This approach is utilized for subgoal generation based on input language instructions while taking into account the limited GPU memory. Additionally, we also rely on large language models to address ASR failures. We tested different methods that account for the presence of ASR failures, incorporating principles of curriculum learning and continual learning:

(a) Data Collection with human and GPT-4. We first manually annotated subgoal generation data from real dialogue history. After collect around 5,000 real user utterances, we let our teammates write subgoals and correct any potential ASR errors in the utterance. For instance, for the instruction "go to the break room and find the bowl", we collect the ASR failure example as "go to the brick room and find the bow" labeled the subgoal as "go to break room, find bowl." As a result, the annotation result becomes the initial component of our subgoal generation data and ASR failure data. Then, utilizing the publicly available trajectory dataset [3], we augment the data with GPT-4 [11] API for the reason that the human annotation is insufficient. We provide the prompts we used in Table 1 and Table 2.

(b) Continual Learning [9] with Combined Data. We train the PyThia model [1] using purely manually labeled data initially, and then utilize GPT-4 to generate ASR failure data, applying curriculum learning principles during the training process. The model is first exposed to simpler, generated

Prompt template	Make the instruction into subgoals formed by verb and noun pairs. The verb must come from this list: Freeze, Pickup, Open, Close, Break, Place, Pour, Toggle, Fill, Clean, Goto, Scan. The noun must be choosen from this list: fridgeupper, cutting board, door, missionitemholder, computer, handsaw, broken radio, dartboard, extinguisher, controlpanel, sodacan, cartridge, cable, carrot, fork, warehouse boxes, color changer, security button, printer hammer, clock, coffeemug, laser toy, bread, red color changer button, pinboard, keyboard, coffeebeans, banana, switch, lasertip, milk carton, printer mug, roboticarm, apple, watercooler, toast, toaster, gravitypad, jar, chair, warningsign, green colorc hanger button, circuitboard, eacmachine, stickynote, coffee maker, teslacoil, cake, candybar, record, bowl, cabinet, spoon, table, lid, drawer, vendingmachine, forklift, fruitpie, firealarm, time machine, burger, bookshelf, fan, poweroutlet, printer figure, hammer, fridgelower, desk, papercup, shelf, puddle, trashcan, cerealbox, floppy, radio, plate, actionfigure, cord, cable, sink, knife, brokencoffeemug, printer, lever, fusebox, broken floppy, trophy, counter, embiggenator, blue color changer button, tesla coil, coffee pot, whiteboard, sandwich, brokenbowl, unmaker, coffee time machine, vending machine button, stool, box, donut, tray, safetybarrier, dart, freezeray, portalgenerator, monitor, laser monitor, freezeraymonitor, laser, virus, notused, screwdriver, microwave, pear, breakroom, small office, manager office, main office, quantum lab, robotics lab, robot lab, reception, warehouse. You can also use 'left, right, forwawrd, backward' in the output. here are some examples: instruction: search the table to deliver. subgoals: goto table, place table. instruction: turn on coffee maker. subgoals: toggle coffee maker. instruction: break plate with hammer. subgoals: pickup hammer, break plate. instruction: pour milk into the cup. subgoals: pickup milk carton, pour cup. instruction: turn right face toward the table. subgoals: goto right, goto table. instruction: freeze the coffee pot with freeze ray and pick it up. subgoals: freeze coffee pot, pickup coffee pot. instruction: approach the table with the laser canon computer. subgoals: goto laser monitor, goto table. instruction: use the time machine to repair the broken mug. subgoals: open time machine, place mug, close time machine, toggle time machine. instruction: turn on the pipe and fill the mug with it. subgoals: pickup mug, toggle sink, fill sink. Instruction: fill the sink with water, then the mug. subgoals: toggle sink, pickup mug, fill sink. instruction: clean the plate in the sink. subgoals: toggle sink, clean sink. Here is the instruction that need your help: Instruction: $\{Instruction\}$. subgoals:
Instruction	walk straight, make right, approach 3-d printer, press red button.
Subgoals (Output)	goto forward, goto right, goto printer, toggle red color changer button.

Table 1: An example of prompting GPT-4 [11] to generate training data of subgoal generation.

ASR failure data to learn basic concepts and patterns, and gradually, we introduce more complex and diverse ASR data, as well as human-labeled data, to refine and expand the model's understanding. Furthermore, we include the subgoal generation data, allowing the model to progressively learn and adapt to the more difficult subgoal generation tasks, even in the presence of ASR failures. Incorporating continual learning [9] with combined data, the PyThia model is continually updated and fine-tuned as it is exposed to new data, ensuring that it remains flexible and capable of handling a wide variety of scenarios, including those with ASR failures. The results of the three approaches are shown in Table 3, with generated ASR failure data and curriculum learning achieving the best performance.

2.2 Image Understanding

Method. Image understanding is a crucial component in the task of embodied AI for robot task completion, as it enables the agent to perceive and recognize objects within its environment. By accu-

Prompt template	I want to gather some data about Automatic Speech Recognition (ASR) errors. There cloud be some homophone involved, for example, pick up a bowl could be mis-recognized as pick up a ball. So when you say $\{Instruction\}$ is it likely to be mis-recognized by ASR? If yes, list the possible wrong versions with number indexes resulting from ASR failures and do not further explain, otherwise just say no.
Instruction	walk straight, make right, approach 3-d printer, press red button.
Subgoals (Output)	1. walk straight make right approach threed printer press read button, 2. walk straight make right approach threed printer press red but in.

Table 2: An example of prompting GPT-4 [11] to generate training data of corecctiong ASR failure.

Method	Exact Match
Manually Labeled Data	51.4%
Generated ASR Data with Curriculum Learning	63.2%
Combined Data with Continual Learning	49.7%

Table 3: Exact match results of the three methods on validation and test datasets.

rately identifying objects and their locations, the agent can better navigate through the environment and interact with the necessary objects to accomplish the given tasks.

Our system utilizes a comprehensive object detection and segmentation framework, called Mask DINO [7]. This framework expands upon DINO [14] by incorporating a mask prediction branch that accommodates all image segmentation tasks, such as instance, panoptic, and semantic segmentation. Mask DINO leverages query embeddings from DINO to predict binary masks through dot product operations on high-resolution pixel embedding maps.

Implement Details. To assess the performance of the model, we fine-tuned the pre-trained SwinL backbone on the same dataset settings mentioned in Alexa Arena [3]. We preprocessed object classes into 86 semantic categories, including a background class. After filtering out objects with an insufficient number of pixels, the dataset comprises approximately 830k training images and 110k validation images. The vision model was trained for 35k steps using a learning rate of 0.00125, a weight decay of 0.001, and the AdamW optimizer. A global batch size of 8 was employed, and the model was trained on 4 Tesla V100 GPUs over a period of three days.

Experiment Settings and Analysis. We employed the same evaluation metrics as in Alexa Arena [3], using the standard COCO evaluation metric, Mean Average Precision (mAP), to evaluate the instance segmentation model. The mAP metric is computed by averaging the precision at various Intersection over Union (IoU) thresholds, ranging from 0.5 to 0.95 in increments of 0.05. The predicted instance class score is disregarded, and a cap on the maximum number of detections per image is applied (set at 100). In order to select an operating point for our end-to-end model and optimize recall, we supplemented the standard COCO mAP metric with a score and IoU thresholded precision metric, refered as t-mAP. This metric is determined by thresholding the output instance class scores at [0.05, 0.1, 0.3, 0.5, 0.7] and thresholding the IoUs at [0.1, 0.3, 0.4, 0.5, 0.75, 0.8] for all 30 (score, IoU) combinations before averaging. We report both metrics for all objects, as well as metrics categorized by object size (small, medium, and large). The results can be found in Table 4 and visualization can be found in Fig 2.

Compared to the baseline model, Mask RCNN, the Mask DINO outperforms mAP for all object sizes. For small objects, Mask DINO achieves an mAP of 62.88 compared to Mask RCNN's 37.63, indicating a 67% increase in performance. Similarly, in the case of medium objects, Mask DINO significantly outperforms Mask RCNN with an mAP of 88.77 as opposed to 60.41, illustrating a 47% improvement. Furthermore, Mask DINO excels in the large object category with an mAP of 95.91 compared to Mask RCNN's 64.72, representing a 48% enhancement in performance. Examining the t-mAP values, we can observe a similar trend of Mask DINO outperforming Mask RCNN across all object sizes. For small objects, Mask DINO achieves a t-mAP of 90.55, while Mask RCNN reaches

	Small		*Medium*		*Large*		*Overall*	
Model	mAP	t-mAP	mAP	t-mAP	mAP	t-mAP	mAP	t-mAP
Mask RCNN [3]	37.63	**91.49**	60.41	92.15	64.72	84.91	46.03	89.50
Mask DINO	**62.88**	90.55	**88.77**	**94.25**	**95.91**	**94.73**	**75.17**	**93.17**

Table 4: Results in percentages on the validation sets. For all metrics, higher is better. All metrics are shown in percentages (%).

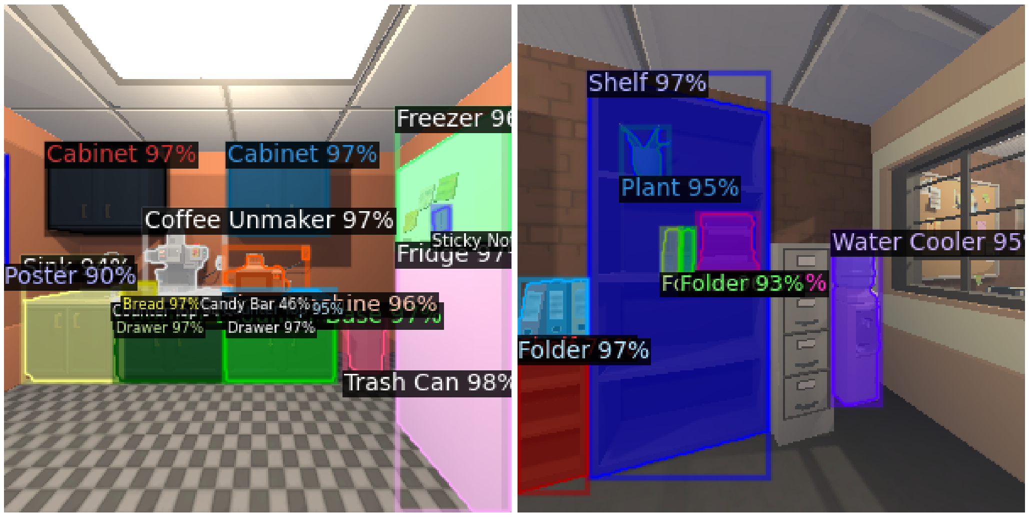


Figure 2: The visualization of the results from Mask DINO.

91.49, showing a slightly lower performance for Mask DINO in this specific metric. However, in the medium and large object categories, Mask DINO surpasses Mask RCNN with t-mAP values of 94.25 (vs. 92.15) and 94.73 (vs. 84.91), respectively. Consequently, Mask DINO proves to be a more effective and robust model for object segmentation tasks.

2.3 Multimodal Understanding

Vision and Language models pre-trained on extensive image-text pairs have demonstrated significant potential in multimodal representation learning [8, 12]. Among these models, CLIP [12] benefits from a curated dataset of 400 million examples and utilizes various prompt templates, enabling it to conduct zero-shot image classification with strong performance. Consequently, we can apply CLIP for instance-level missing object class identification and self-correction, considering that the object detection module in the pipeline can yield incorrect predictions. We show the example of missing object class identification and self-correction in Figure 3.

Self-Correction. We utilize CLIP to discern objects within a single image. For example, given the imperfect recognition accuracy of vision model, multiple distinct broken bowls may be detected in a sentence such as "pick up a broken bowl." To address this issue, we calculate the similarity score between the target object's name and the object region in the image. We mask out image regions not included in the bounding box for this purpose. Subsequently, the two object regions with the highest similarity to the target object's name are returned. We then request user confirmation to identify the target object by asking whether the left or the right object is the intended one among the returned pair. The confirmed object is subsequently outputted as the detected region.

Missing Objects. CLIP is also applicable when no target object is detected in the current image. We compute the pairwise score between the object region in its current bounding box with its predicted object name and the object region in its current bounding box with the target object name recognized from the user's current round instruction. If the score for the target object name is higher, the name is

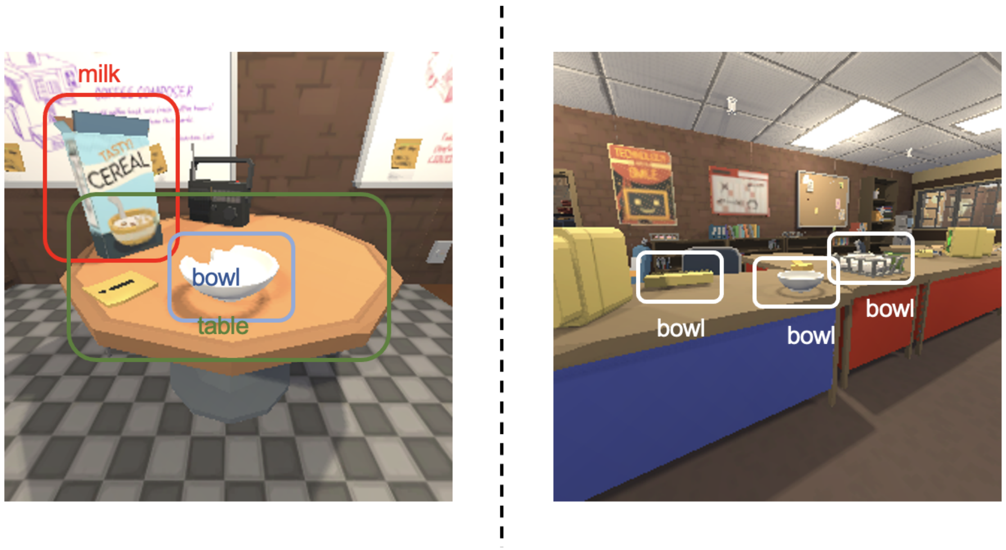


Figure 3: The examples of missing object class identification and self-correction. The left figure shows the case when the vision model wrongly predict the cereal to be the milk and the current instruction is asking the agent to "find the cereal". The right figure shows the case when two unrelated objects are also predicted to be the target object "bowl". We apply CLIP for missing object class identification and self-correction.

stored in a dictionary. Upon traversing all bounding boxes, we retrieve the name with the maximum score from the dictionary and update the name to be the target object name. For example, for the target object "milk", the pretrained vision model tends to falsely recognize the "cereal box" object as milk in the current image. We thereby traverse all the detected object regions in the current image and correct the prediction as the milk region has the highest score with the text label "milk".

3 Knowledge Module

Besides the perception mode, the system also depends on common sense knowledge, just as humans require in the real world, to enhance their comprehension and engagement with objects. In Slug-JARVIS, the knowledge could be from the existing knowledge corpus (**offline**), or be accumulated during the user interaction process (**online**).

3.1 Offline Knowledge

Useful information is extracted from the given dataset, which comprises the summary and dialogue of tasks, subgoals, and corresponding images. The subgoals consist of action-object pairs, such as "Pickup Cake". The dataset provides information on object affordance, object alias, and task subgoals, allowing for the extraction of two levels of common sense: object common sense and task common sense.

Object common sense includes object affordance, which refers to the corresponding action of an object, and object alias, which is extracted from the dialogue. For example, the object affordance for an apple would be picking it up and placing it, while the object alias for a computer may be a monitor. Object common sense is used to prompt the user for appropriate actions based on the object they are interacting with. For instance, if the user is interacting with an apple, the system will suggest picking it up, while attempting to fill it would prompt a correction based on the object affordance. Additionally, object common sense also includes the synonymous names for each object since they might be called different names by users.

Task common sense is extracted from the task subgoals and refers to a series of subgoals that prompt the user for the next action based on the current state. For example, in the task "3D Printing the

Action Figure", if the simbot has picked up the action figure cartridge with the power on, the system will prompt the user to place the cartridge into the printer based on the recorded task common sense. Both levels of common sense are derived from the dataset, allowing the system to adapt to various environments.

3.2 Online Knowledge

As various tasks and environments exist, updating the knowledge base and leveraging each other's experiences presents a challenge. Our observations indicate a strong correlation between tasks and environments, suggesting that environmental features can assist the agent in identifying current tasks. Additionally, execution sequences can aid the agent in determining which environmental knowledge from others can be applied to current tasks. Therefore, we introduces two methods below for task identification and knowledge updating.

Progressive Task Retrieval. The initial step in updating and utilizing the task-specific knowledge base involves recognizing the current tasks. To this end, we have developed a progressive task retrieval method that enables the agent to identify the ongoing task using image and subgoal sequences. For image information, we first encode the observed image from the initial K steps into embedding vectors. These vectors are then employed to calculate the cosine similarity with the existing database. If the similarity falls below a threshold or the database is empty, we classify the current task as unknown and establish the current embedding as a new identifier. Conversely, if the similarity is above the threshold, we have found the matching task. Among the unknown tasks, some are misclassified due to random noise in the image observations. To address this, we have devised a span-matching algorithm that compares current sequences to previously recognized sequences at the subgoal level. If a common span exists between the current and target subgoals, we consider the unknown task to be the target task.

Task-specific Knowledge Updating. Despite agents executing different sequences across various interactions, we anticipate the extraction of common task-specific knowledge that can be further utilized by others. This common knowledge primarily consists of two aspects: semantic maps and subgoals.

For the semantic map, the agent records every observed object during interactions. The semantic map only contains sparse object information, including object class, score, bounding box area, observing pose, and the nearest viewpoint. When two object instances are observed from the same viewpoint, only the object with the higher score is preserved on the map. Objects can be classified into two categories: large objects and small objects. Large objects serve as environmental landmarks and remain stationary, allowing their locations to be shared with others. Conversely, small objects can be picked up and relocated, making only their initial positions useful for subsequent interactions. After each step, the agent updates the semantic map associated with the current task, which is later employed during the initial step for new users.

While tasks may involve varying action sequences, they are likely to share some common subgoals, particularly interaction subgoals such as picking up or toggling. As a result, we extract subgoal sequences from previous successful experiences and incorporate them as potential subgoals for current tasks. When a user starts a new game, this subgoal information can be used as hints if the user encounters failure multiple times.

4 Task Execution Module

The existing understanding module effectively processes users' intents into subgoals; however, a discrepancy persists between subgoals and executable actions. To address this challenge, we introduce a hierarchical task manager, shown in Fig. 4, encompassing a subgoal manager and an action manager. During the inference phase, the subgoal manager integrates external knowledge with subgoals to generate an action list. Subsequently, the action manager translates actions from the action list into executable actions. We shall elucidate these components in greater detail in the subsequent sections.

Knowledge-Augmented Subgoal Manager. As delineated in Section 2, the understanding module translates utterances into subgoals, formatted as "[action, object]" pairs. During the inference phase,

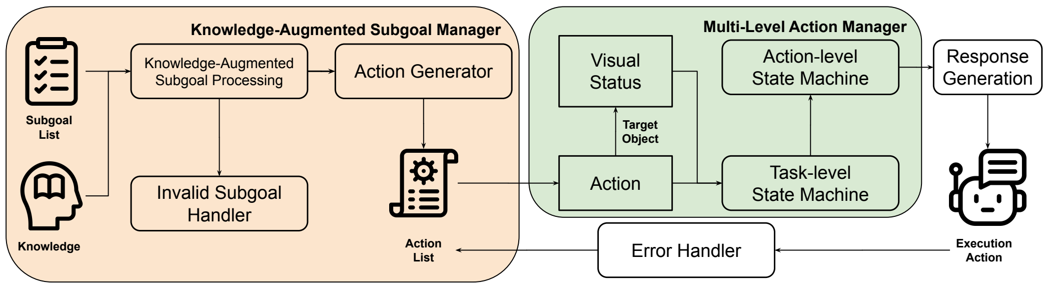


Figure 4: **The structure of task manager.** The image includes both the subgoal manager and action manager. The subgoal manager can post-process the generated subgoals by using external knowledge and convert them into action lists. The action manager will process the action with visual status and two-level state machines for the final executable output.

the agent extracts subgoals sequentially from the list, individually. Nonetheless, not all subgoals warrant conversion into actions – for instance, the agent should refrain from picking up an additional object if one is already in hand. Consequently, incorporating external knowledge for subgoal post-processing is crucial. This knowledge comprises both task-agnostic and task-specific information, introduced in section 3. Four subgoal-checking procedures are incorporated: (a) Format checking. The subgoal must include both the action and target object. If either component defaults to the agent's default value (None), the subgoal is deemed an invalid input. (b) Affordance checking. As most objects cannot accommodate all actions (e.g., an apple cannot be opened), discerning valid combinations is essential. Thus, the commonsense object affordances are employed in this check. (c) Object status checking. The proposed subgoal may conflict with the current object and robot statuses. For example, when an agent must place an object into an openable receptacle, the agent must hold the object, and the receptacle must be open. To address such issues, we devise logical predicates for rules. For example, the aforementioned rule can be expressed as $Place(object, receptacle) = isPicked(object) \wedge isOpened(receptacle)$, where $isPicked(object)$ holds true if and only if the agent has grasped the object, and $isOpened(receptacle)$ holds true if and only if the agent has opened the receptacle. (d) Object location checking. As the agent can construct a semantic map during exploration, verifying the proximity of the target object to the current position is essential. If the object is not nearby, the agent must first relocate to the relevant position. All invalid subgoals will be sent to the invalid subgoal handler, and the handler will return appropriate responses according to the error type. Other valid subgoals will be translated into the action list. Since the subgoal can be a semantic combination of atom actions, the action list usually is larger than one. For example, each interaction subgoal can be translated to goto and interact with the target object.

Multi-Level Action Manager. The action list generated typically comprises multiple actions, necessitating that the action manager consider individual actions across various rounds. Consequently, the manager incorporates a two-tiered state machine. The first tier, the task-level state machine, examines the agent's behavioral patterns over several rounds and encompasses five distinct states: Normal, WaitForChoose, DecideAfterChoose, WaitForConfirm, and DecideAfterConfirm. The second tier, the action-level state machine, reflects different agent statuses within a single round and consists of five states: Executable, LookAround, TargetNotFound, RightLeftSelection, and NumberSelection.

During the inference phase, the action manager extracts an action from the list and, if necessary, attempts to locate the target object. Regarding visual status, three potential scenarios exist: NotFound, OneFound, and MultiFound. The visual status, user utterances, and current action type collectively influence the task-level and action-level state machines. The default task-level state is Normal. When the visual status is OneFound, indicating that a single target object instance has been identified, the action-level state becomes Executable, signifying that the current action can be executed directly. If the visual status is NotFound, indicating that no target has been located, the action-level state is initially set to LookAround, prompting the agent to explore the surroundings. If the target remains undetected, the action-level state transitions to TargetNotFound, guiding the agent's appropriate response. Moreover, if the visual status is MultiFound, signifying the detection of multiple target objects, the task-level state must shift to WaitForChoose, while the action-level state transitions to

Dialog Act	Intent	Example
`ClarifyInstruction`	Task info	I think you want to pick up something, can you repeat that again?
`AskForInstruction`	Task info	I have picked up the mug, what next?
`CanNotPerform`	Subgoal Completion	I can not execute
`ObjConfirm`	Subgoal Completion	I detected multiple apples, which one are you talking about, the left one or the right one?
`FindConfirm`	Subgoal Completion	There are several places to search in the current room. Do you want me to keep searching?
`WelcomeMessage`	Inform	This is a game in that I do what you say to accomplish some task.
`PromptSticky`	Inform	You can try reading the sticky note to get more information about the game.
`SelfCapability`	Inform	I am designed to help you finish the game. Can you tell me the next step?
`LowLevelAction`	Inform	If you see the target, you can help me go to it using actions like go forward, turn right.
`GameInfomation`	Inform	The task goal is on your top-left screen. The map of the environment is in the top right.
`CommonsensePrompt`	Inform	Maybe you can try to pour the milk in the bowl?
`ActionToPerform`	Trust & Satisfaction	I will pickup the bowl.
`ActionPerformed`	Trust & Satisfaction	I poured the cereal into the bowl.
`Acknowledgement`	Trust & Satisfaction	Great!

Table 5: Dialogue acts, intents, and examples for response generation.

RightLeftSelection or NumberSelection, contingent on the number of candidates. The agent will then refrain from extracting subgoals from subsequent utterances and instead attempt to discern the user's selection intent. Subsequently, the task-level state will transition to DecideAfterChoose and ultimately return to Normal. Furthermore, if an action list becomes excessively lengthy, the task-level state will change to WaitForConfirm after several steps, indicating that the agent requires user confirmation to proceed with the action list. Upon receiving the intent, the task-level state will transition to DecideAfterConfirm and ultimately return to Normal. Lastly, the NLG and error handler submodules supply the appropriate dialogue values for both successfully executed and failed actions.

5 Goal-oriented Response Generation Module

During the process of complex task completion, the embodied agents may lack the common sense or domain-specific knowledge that is required to finish the task or do not know the exact goal of the task. Therefore, the ability to initiate dialog with humans and get important information from the dialog is crucial for embodied agents to perform complex tasks collaborating with humans [4, 6]. In the embodied environment of Arena [3], the embodied agent is not aware of the task goal, which is given to the human player in the form of natural language instruction. The player will give instructions to the agent, and the agent is required to perform complete tasks according to the player's instructions. In this setting, the agent needs to generate different responses to the player in different scenarios with different intents. To better organize the response generation, we classify response generation based on the intents and dialogue acts [6, 13] as in Table 5.

First, the agent needs to ask the user for the next instruction appropriately to get more task information. Second, during the subgoal completion process, we may need help from the player to complete one subgoal. Third, the player may not be an experienced player and is not sure how to give instructions to the agent efficiently, and the agent needs to inform the player about the information of the game in this case. Lastly, besides task completion, during the human-agent collaboration process, the agent also needs to gain human trust and increase human satisfaction. Therefore, the last important response category is the responses to gain human trust and improve human satisfaction.

Request for task information. A task in the Arena game session is usually composed of several subtasks. Therefore, the player needs to give several instructions to the agent during the game. As introduced in Fig 1, after the agent receives an instruction, it will plan and execute several subgoals accordingly. During this process, it mainly needs to ask the user for task information on two occasions. As listed in Table 5, the first situation is when the agent is not sure about the intent of the user due to various reasons and needs to ask for clarifications. In this case, the agent will first recognize what information is unclear in the player's instruction through language understanding. Then, it will provide this recognition as feedback to the player and ask for corresponding information from the user. The second situation is when the agent finishes the execution of one instruction. It will inform the user of completing the current subtask and inquire about the next instruction to complete the task.

Request for help in subgoal completion. During the completion of a subgoal, the agent might be uncertain about which action to choose and may need help from the player in some cases. For example, as shown in Table 5, when the agent detects multiple target objects in a scene and is unsure which one the player is referring to, it can ask the user to make a choice.

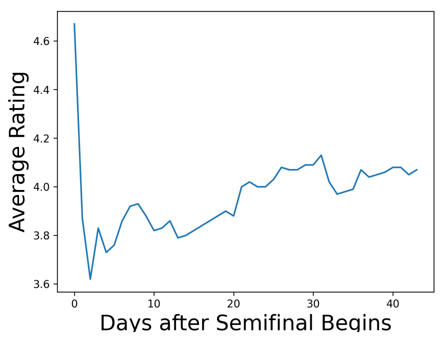
Inform the user of game information. A new user might be confused about the general goal of the game. To detect possible confusion of the players and prompt them appropriately with useful information, we utilize the language understanding of the player's utterance, including out-of-topic detection and unclear instruction detection. For example, as in Table 5, when the agent detects an out-of-topic utterance, it will inform the user about its capability or/and the information for the game. Besides passively prompting the user, we also actively prompt the user on how to play the game via a welcome message like the example in Table 5.

Response for human trust and satisfaction. During the game, how the embodied agent works and completes tasks are unknown to the player. Therefore, the players may not know what the agent is doing and causing distrust, especially when the agent performs a long sequence of actions. To this end, we let the agent be transparent about their actions in mainly two ways: Telling the player when it is going to execute an action and after it executed an action.

Language generation with two-step pipeline. For each category and dialog act, the response needs to be diverse, concise, and informative. Therefore, we follow a generating-filtering pipeline to generate the responses using ChatGPT [10]. In the first step of the pipeline, we will first brainstorm some responses for each situation. Then we input the context and intent of the response (e.g., I am an embodied agent, the user gives an instruction to me to do something on an object, but I miss the action user said. How can I ask the user?) and the responses we brainstormed to ChatGPT as examples. We then let ChatGPT generate several diverse response candidates. Note that we will not use the player's utterance as input. In the second step of the pipeline, we delete or modify the verbose and confusing responses. The remaining responses are the candidate responses for each case, which will be randomly chosen when needed.

Tone management according to sentiment. Besides the content in the response, tone management is also important in human communication. Therefore, we leverage the Speech Synthesis Markup Language (SSML) of Alexa to manage the tone of response according to the sentiment of the text content. Specifically, we manage the tone in two dimensions: emotion and intensity. For emotion, we utilize 'excited' and 'disappointed' for different responses. For intensity, we switch between 'High', 'Medium', and 'Low'.

6 Analysis



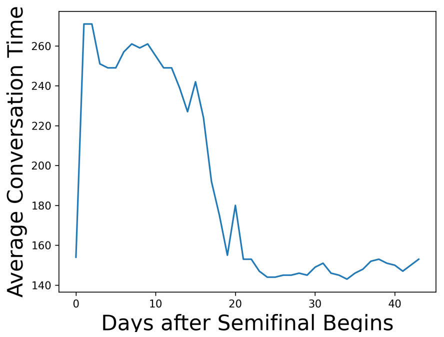
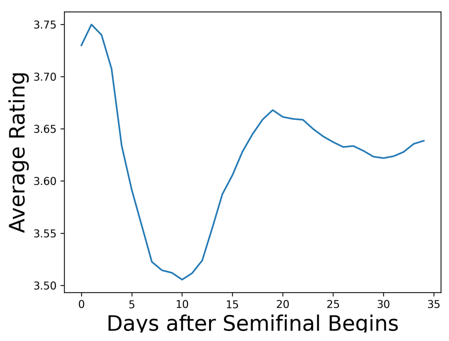


Figure 5: **Performance during semifinal.** (a) Left is average Alexa user rating. (b) Right is the average Alexa user conversation time for each game.

In the present study, the performance of the system was evaluated by examining user interactions and assessing the quality of ratings. For each potential issue encountered, a thorough analysis of the dialogue and potential error messages was conducted to enhance the system's efficacy. As depicted in Figure 5, the average user rating and average conversation duration have been tracked since the commencement of the semifinal stage. During the initial ten-day period, the system's performance was relatively low due to ongoing adaptations to the codebase in response to the new environment. However, as development continued, there was a notable increase in the score, then it stabilizes at approximately 4.07. As in Figure 6, new games were introduced and there is a core drop. Then the



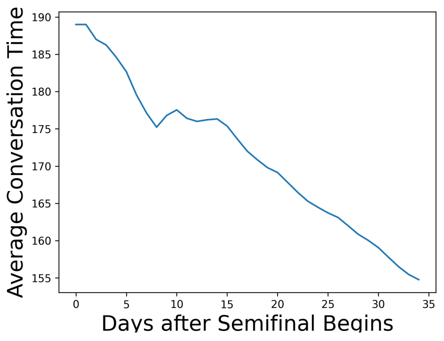


Figure 6: **Performance after the end of semifinal.** (a) Left is average Alexa user rating. (b) Right is the average Alexa user conversation time for each game.

Last Turn	Instruction: Pick up the bowl. JARVIS: Sorry, I did not find bowl in current position. Do you want me to initiate a search?
Static Setting	Instruction: Place it on the table in the breakroom. JARVIS: Sorry, I do not have bowl in my hands. Do you want me to search for it?
Human	Instruction: Yes, please search for it. JARVIS: (after searching and finding it). I found it! What should I do next?

Table 6: An example of our interactive system generated by our team member. Our model successfully executes a task with a real human but fails under the static instruction setting.

average conversation time exhibited a decreasing trend and the user kept improving, indicating that users required less time to complete the game as we keep developing. This improvement can be attributed to the progressive augmentation of the system's knowledge base and the integration of additional logic units throughout the development process.

In the context of the Alexa Prize Simbot Challenge, specifically the instruction-following task, our research group attained a Goal Completion metric of 3.66. This outcome demonstrates a less satisfactory performance when juxtaposed with our achievements in the online interactive setting. A thorough examination of the generated subgoals and final actions elucidated that the performance disparity can be predominantly attributed to the numerous components within the SlugJARVIS framework, which are primarily designed for conversational environments.

The implementation of such a model in an instruction-following setting, where users do not anticipate verbal feedback from the robotic system, inadvertently results in a decline in overall performance. Table 6 provides an example where our system failed under static instruction following setting.

7 Conclusion and Future Work

In this technical report, we demonstrate how to instruct embodied agents in conducting household tasks using dialogue-based interactions on the Alexa Arena simulation platform. Our creation, SlugJARVIS, is a neuro-symbolic embodied agent that utilizes information from multiple modalities to make decisions based on common sense reasoning. We anticipate that the technical advancements presented in this report will further facilitate the development of embodied agent research in both academia and industry.

Based on our framework, there are still many remained challenges in conversational embodied agents. First, there are users with different levels of familiarity of the Arena game with different personal preferences, noticing this during the interaction with the users and make personalized adaptations can further help user experience. Second, how SlugJARVIS collaborate with other agent to jointly finish the game is also an interesting future direction.

References

[1] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[3] Qiaozi Gao, Govind Thattai, Xiaofeng Gao, Suhaila Shakiah, Shreyas Pansare, Vasu Sharma, Gaurav Sukhatme, Hangjie Shi, Bofei Yang, Desheng Zheng, et al. Alexa arena: A user-centric interactive platform for embodied ai. *arXiv preprint arXiv:2303.01586*, 2023.

[4] Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme. Dialfred: Dialogue-enabled agents for embodied instruction following. *arXiv preprint arXiv:2202.13330*, 2022.

[5] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. 2017.

[6] Spandana Gella, Aishwarya Padmakumar, Patrick Lange, and Dilek Hakkani-Tur. Dialog acts for task driven embodied agents. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 111–123, Edinburgh, UK, September 2022. Association for Computational Linguistics.

[7] Feng Li, Hao Zhang, Huaizhe xu, Shilong Liu, Lei Zhang, Lionel M. Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation, 2022.

[8] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975, 2022.

[9] Andrea Madotto, Zhaojiang Lin, Zhenpeng Zhou, Seungwhan Moon, Paul Crook, Bing Liu, Zhou Yu, Eunjoon Cho, and Zhiguang Wang. Continual learning in task-oriented dialogue systems. *arXiv preprint arXiv:2012.15504*, 2020.

[10] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[11] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

[12] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[13] Dian Yu and Zhou Yu. Midas: A dialog act annotation scheme for open domain humanmachine spoken conversations. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2019.

[14] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection, 2022.