
GaichoAI: A Proactive Modular Embodied AI that Collaborates with Humans

Jiachen Li Deepak Nathani Tsu-Jui Fu Xinyi Wang Xifeng Yan
UC Santa Barbara

Abstract

We introduce an embodied AI agent designed to interact with human users, achieve high mission success rates, and deliver a satisfying user experience. Our agent utilizes a modular framework, incorporating explicit state updates based on previous actions, next-action predictions derived from user input, target object detection from observations, and various exploration strategies for locating unseen targets. Moreover, the agent proactively suggests the next steps upon reaching specific states, reducing the need for detailed user commands. We showcase our approach on the Alexa Arena [Gao et al., 2023] platform, achieving an average rating of 4.29 from February 21st to March 21st. These results demonstrate the substantial potential of our embodied AI learning framework to enhance Human-Robot interactions and promote successful task completion across a variety of applications.

1 Introduction

Recent advancements in multi-modal research, especially in large-scale vision and language models [Radford et al., 2021, Ramesh et al., 2022, Rombach et al., 2022, Ouyang et al., 2022] such as Flamingo [Alayrac et al., 2022], GPT-3 [Brown et al., 2020], and GPT-4 [Open AI, 2023], have paved the way for more complex and capable embodied AI (EAI) agents that can effectively collaborate with human users. To fully realize the potential of these techniques and build a versatile human assistant, it is essential to maintain a positive user experience while prioritizing mission success. However, most existing methods [Pashevich et al., 2021, Blukis et al., 2022, Min et al., 2021, Sharma et al., 2021, Jia et al., 2022] in EAI solely focus on following user commands rather than truly interacting with the user to accomplish given missions, partly due to the absence of an appropriate benchmark.

Recently, Gao et al. [2023] introduced the Alexa Arena, a novel EAI platform that supports real-time human-robot interactions, facilitating human evaluation. Inspired by this development, we designed a conversational EAI system that collaborates with human users. Our approach relies on a modular framework that allows the agent to adapt and respond efficiently to dynamic situations. The framework comprises explicit state updates based on previous actions, next-action predictions derived from user input, target object detection from observations, and various exploration strategies for locating unseen targets. This versatility empowers the agent to be proactive in suggesting the next steps upon reaching specific states, ultimately reducing the need for detailed user commands and enhancing overall collaboration.

To validate the effectiveness of our embodied AI agent, we demonstrate its performance on the Arena Run time system, where it achieves an average rating of 4.29 out of 5.0 from February 21st to March 22nd, a period when all teams will release their best models to compete for the final qualifications. These promising results illustrate the significant potential of our learning framework for improving human-robot interactions and promoting successful task completion across a wide range of applications. In the following sections, we will discuss the components of our modular framework before delving into the results and potential future directions.

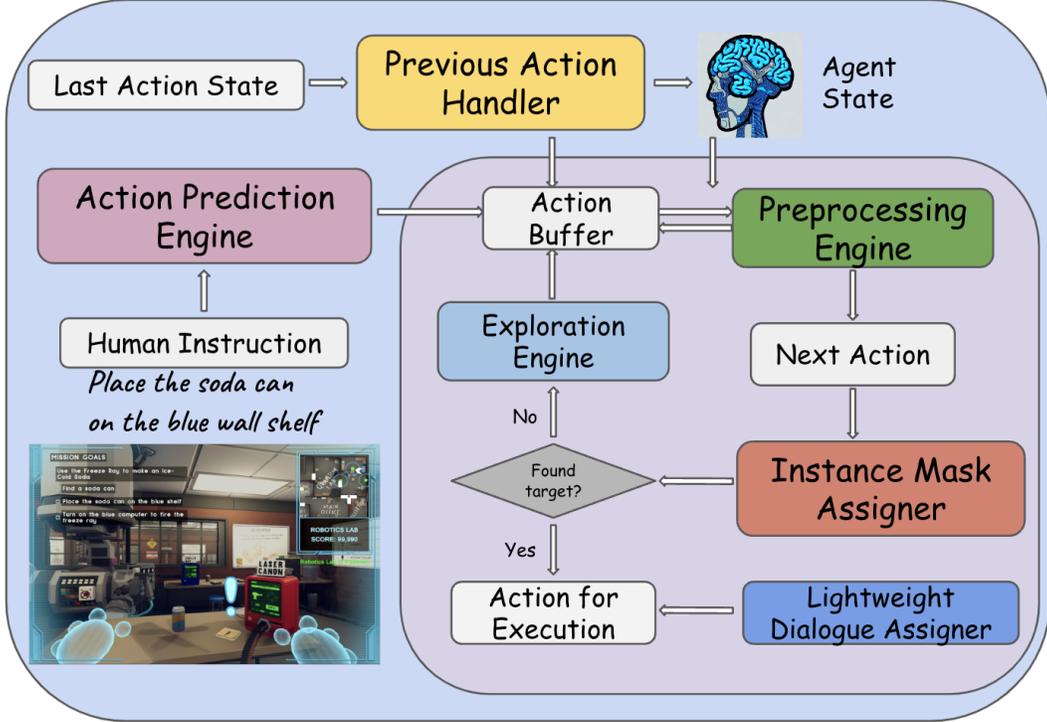


Figure 1: The system design of our GauchAI, a modular EAI agent. Given a new human instruction, the Action Prediction Engine predicts a sequence of actions from the human instruction and populates the Action Buffer. Given an action prediction request from the Arena Runtime platform, the Previous Action Handler extracts the Last Action State to update the agent state. Moreover, it may also expand the Action Buffer with new action sequences based on the agent’s states and the Last Action State. Next, a candidate action is popped from the Action Buffer and passed to the Preprocessing Engine, which accepts or rejects the candidate action based on the current agent state. For example, if the candidate action is to place the soda can on a wall shelf while the agent is not holding the object, the Preprocessing Engine will reject the action and add a `<Pickup, Soda Can>` to the top of the Action Buffer. For accepted candidate actions, the Instance Mask Assigner tries to predict an instance segmentation mask from the RGB images for the target object. If the target object cannot be detected from the current observations, the Exploration Engine takes over and generates an action sequence to search for the object. Otherwise, the action will be sent for execution with a lightweight response generated by the Lightweight Dialogue Assigner. Note that all modules take RGB images as inputs. For simplicity, we have omitted the arrows between modules.

2 GauchAI: A Modular EAI Framework

Our GauchAI system, as illustrated in Figure 1, consists of a Previous Action Handler, an Action Prediction Engine, a Preprocessing Engine, an Instance Mask Assigner, an Exploration Engine and a Lightweight Dialogue Assigner.

The Action Prediction Engine generates new action sequences in response to human instructions, taking into account the current state of the agent and the objects in the environment. The Previous Action Handler updates the agent’s inventory and environment state. All actions are first processed by the Preprocessing Engine before being executed in the simulation environment. Once the Preprocessing Engine accepts an object interaction action, the Instance Mask Assigner will assign an object mask to the target object based on the output of an instance segmentation model. Additionally, if the Instance Mask Assigner cannot detect a target object in the current RGB observations, the Exploration Engine is used to locate it. To increase user experience, we implement a Lightweight Dialogue Assigner to generate lightweight dialogue for object interaction actions. In the rest of this section, we will elaborate on each module.

2.1 Action Prediction Engine



Figure 2: Human instruction: take the milk from the fridge to the red wall shelf in the Robotics Lab. The expected action sequence requires going to the fridge, opening the fridge, picking up the milk, going to the robotics lab, and finally going to the red wall shelf.

The Action Prediction Engine employs a language and visual-based approach to predict action sequences based on human instructions. The engine populates the Action Buffer by utilizing a rule-based method to parse user utterances, identifying the action name and target object based on the affordances of the objects. Specifically, for the 11 object interaction actions and navigation actions supported by the Alexa Arena, the engine predicts the target object based on the action name, considering the object’s characteristics. Additionally, the engine utilizes an instance segmentation model from the Instance Mask Assigner to obtain the set of object classes present in the current observed RGB images. This visual information is useful for addressing scenarios where the user’s intended action is under-specified, such as when the user omits the target receptacle when instructing the action to *put down the bowl*.

Even though the action execution in the Alexa Arena only requires predicting the target object, we propose defining the data structure of the action as a tuple of three elements given by

$$\textit{Action name}, \quad \textit{Source Object}, \quad \textit{Target Object}$$

While the *Action name* is as simple as a string, each *Object* contains four fields, including *name*, *color*, *state*, and *anchor object*. Notably, the *anchor object* is utilized for location. The *Source Object* refers to the object expected to be held when executing the action $\langle \textit{Action Name}, \textit{Target Object} \rangle$. Next, we will conduct a case study to illustrate the definition of each item using the instruction below.

take the milk from the fridge to the red wall shelf in the Robotics Lab.

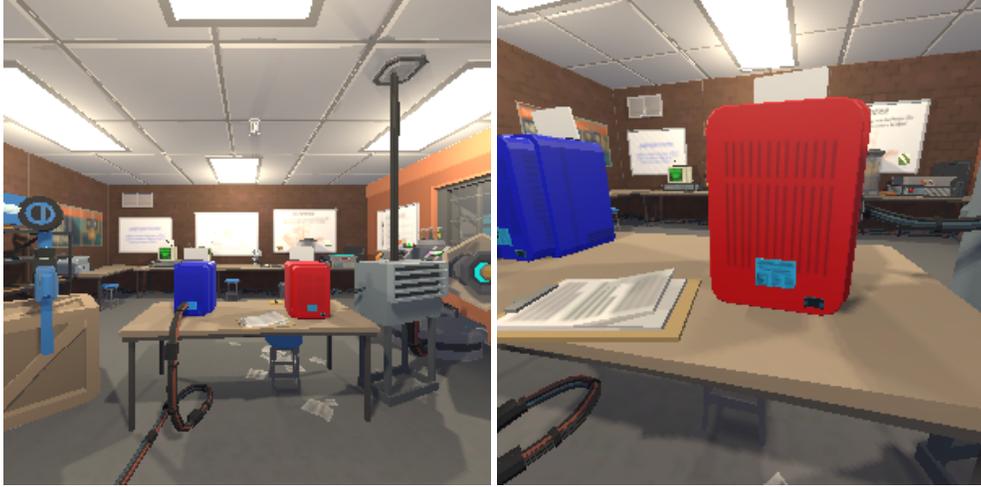


Figure 3: Left: A box is blocking the way. Right: Execute a $\langle Goto, Computer \rangle$ command cannot go to the other end of the computer, thus preventing the agent from activating the laser.

The expected output of our Action Prediction Engine is given by

```

Action name = Goto
Source Object = {
  name: milk,
  color: None,
  room: None,
  anchor object : {name: fridge, color: None, room: None, anchor object: None}
}
Target Object = {name: wall shelf, color: red, room: Robotics Lab, anchor object: None}

```

Therefore, the predicted sequence of actions involves going to the fridge to pick up the milk, then carrying the milk to the Robotics Lab, and finally arriving at the red wall shelf as shown in Figure 2. In Sec. 2.3, we will further demonstrate the effectiveness of this representation for action prediction when combined with the Preprocessing Engine.

In addition, there are scenarios where our Action Prediction Engine can only predict either the action name or the target object name due to Automatic Speech Recognition (ASR) errors. To address this, we have designed our agent to kindly request a rephrasing from the user. Specifically, when the agent can successfully predict the action name, we request the user to repeat the target object name. Conversely, when the object is predicted but missing the action name, we have fine-tuned a BART [Lewis et al., 2019] model that takes into account the agent’s inventory and predicted object name to predict the missing action, given that the affordance for an object is limited. For instance, when the agent is holding a hammer facing a breakable object, the intended action will likely be *break*. Moreover, we confirm the predicted action with the user to prevent unintended action execution, enhancing the reliability and accuracy of the system, and thereby improving the user experience.

2.2 Previous Action Handler

As the action prediction request from the Alexa Arena Runtime System comes with the Last Action State, we propose the Previous Action Handler with two primary functionalities.

- Update the agent state and the belief about the environment when the last action is executed successfully. When reaching certain states of the agent and the environment, proactively recommend the possible next steps to the human users.
- Handle failed action executions, including providing intuitive error messages to the user and possibly proposing the next actions to recover from the error.

Specifically, we will record the inventory of the agent. For example, the inventory gets updated to *milk* after acquiring the milk from the fridge and reset to empty after the agent puts the milk on the table. Moreover, we will also record the state of objects in the environment, including the receptacle of a pickupable object, whether a receptacle with a door is currently opened, and the filling status of a container. We emphasize this information is extremely important, as the user may under-specify their intended actions, which could lead to errors in the system. As is shown in Figure 2, the user omits the instruction to open the fridge before picking up the milk. As the fridge is not opened by default, our agent can thus infer that it should first open it to locate the milk during action execution.

In addition to recording the agent and environment states, the Previous Action Handler is designed to proactively suggest possible next actions to users based on certain contextual cues. For instance, when the agent arrives at the red wall shelf or the microwave with a heatable object, such as a cake, it is highly probable that the user intends to heat the cake. Similarly, when the user arrives at the 3D printer with a printer cartridge, it is very likely that the user plans to print an object with the printer. In these scenarios, the Previous Action Handler will ask the user for confirmation on whether to execute these actions, reducing the user’s effort in providing detailed instructions to complete the task. This design significantly enhances the efficiency and effectiveness of human-robot interactions, providing an improved user experience.

In addition to suggesting the next actions, we have found it extremely beneficial to generate encouraging responses when users successfully achieve certain goals. Although we do not have access to pre-defined subgoals of a gameplay mission, we can still provide verbal praise to users when they successfully operate a machine or find a previously undetected target object. To generate these encouraging responses, we first gather all objects with the TOGGABLE property in the Alexa Arena environment and draft the responses based on the functionalities of the TOGGABLE object. Take the time machine that is designed to repair broken objects as an illustration. If a user successfully repairs a broken bowl using the time machine, our system generates an encouraging response such as

Brilliant! Now the bowl has been repaired!

To add more variety to the responses, we also use ChatGPT [Open AI, 2023] to generate alternative phrasings, populating the set of candidate responses. During inference time, we randomly sample one response from the candidates. We refer interested readers to our source codes for the full picture of our conversation strategies.

There are also scenarios when the last action cannot be executed successfully. However, some of these actions can be resolved automatically. For instance, when the error type is *TargetOutOfRange* and the agent is executing an object interaction other than *Goto*, our Previous Action Handler will insert a *<Goto, Target Object>* action before re-executing the last action. This approach eliminates the need for the user to provide additional commands to resolve the issue, resulting in a more seamless interaction. However, it is essential to note that the key is to avoid looping action execution, which can result in undesired outcomes.

There are cases where the previous strategy fails, such as when a box blocks the way for the agent to activate the laser monitor in the robotics lab. As shown in Figure 3, the agent cannot execute the previous action without first removing the box with the robotics arm. Therefore, the agent can approach and turn on the computer only after removing the box, demonstrating the importance of considering the environment’s physical constraints and limitations when designing the Previous Action Handler.

2.3 Preprocessing Engine

The primary objective of an EAI agent is to work collaboratively with a human user by following their instructions. However, human instructions are often under-specified because of our common sense of the world or visual cues that are strong enough to infer the missing information. As shown in Figure 4a, we might issue the command "put down the bowl" without explicitly specifying the target surface, such as the table, because the agent is already in front of it or because it’s the only available receptacle in the field of view. Additionally, even when the agent can predict the action and object name perfectly from the sentence, it may still execute a different action from the user’s intention. For instance, in Figure 4b, the word *table* can refer to any object with a flat surface. However, since common sense is not incorporated into our Action Prediction Engine, it cannot generate the desired action without pre-processing the information before executing the actions from the Action Buffer,



(a) put down the bowl.

(b) place the cake on the table.

Figure 4: (a) Human user may provide under specified command that omits the table as the target receptacle. (b) Human may treat the red wall shelf as a flat surface, thus issuing a command to *place the cake on the table*.

which highlights the importance of the capability to reason based on contextual information and domain knowledge, which can lead to more effective collaboration between humans and machines.

Motivated by these challenges, we have integrated a Preprocessing Engine into our GauchAI framework, as shown in Figure 1. The Preprocessing Engine helps us to decide whether to accept or reject a candidate action from the Action Buffer and is also where we inject the most human prior into our EAI system. Specifically, the Preprocessing Engine checks whether the *source object* mentioned in Section 2.1 is in the agent’s inventory. If the source object is not present, the Preprocessing Engine adds a *Pickup* action to the Action Buffer before executing the candidate action. Similarly, when placing an object into an openable receptacle, such as the time machine and microwave, the Preprocessing Engine automatically adds an "Open" action if these machines are closed.

In essence, the Preprocessing Engine can be considered an extension of the Action Prediction Engine, with access to the most recent states of the agent and the environment. Although these two modules currently operate in a rule-based manner, we believe that as more data is collected from human-robot interactions, it will motivate the development of more powerful learning-based methods. Therefore, we see the Preprocessing Engine as a critical step toward the ultimate goal of creating intelligent EAI systems that can work seamlessly alongside humans.

2.4 Instance Mask Assigner

To identify the target objects in the observed RGB images, we employ an Instance Mask Assigner to predict an instance segmentation mask from the RGB images for the target object(s). In addition to the provided vision model based on MaskRCNN He et al. [2017], we fine-tune a MaskFormer Cheng et al. [2021] on the provided vision data, achieving 41.8% mAP for small objects, 85.6% mAP for medium objects, and 94.5% mAP for large objects on the validation sets. This significantly outperforms the v1 place-holder model, which had 73% mAP for medium objects and 72.2% mAP for large objects.

However, the instance segmentation module alone is insufficient to correctly identify the target objects, especially when there are multiple instances of the same object class in the observations. To overcome this limitation, we employ CLIP Radford et al. [2021] to identify the best instance that aligns with human instruction. Figure 5 outlines the process we follow to obtain input for the CLIP visual encoder. In this specific example, we try to classify whether or not the detected computer is infected. We use the following prompts as the input to the CLIP text encoder

- 1) a skull on the screen.
- 2) a screen with block.
- 3) a screen with paragraphs.

Note that we use three prompts as different normal computers can display various content. Nonetheless, we classify a computer as infected only when its visual feature has the highest similarity score to the *first* prompt. It’s important to note that our approach to determining the computer’s state is imperfect. In situations where the agent faces the back of the computer, as shown in Figure 3, it may be impossible to determine the computer’s state accurately. Therefore, while our approach works well in many cases, it’s important to remember that it may not be infallible in every situation.

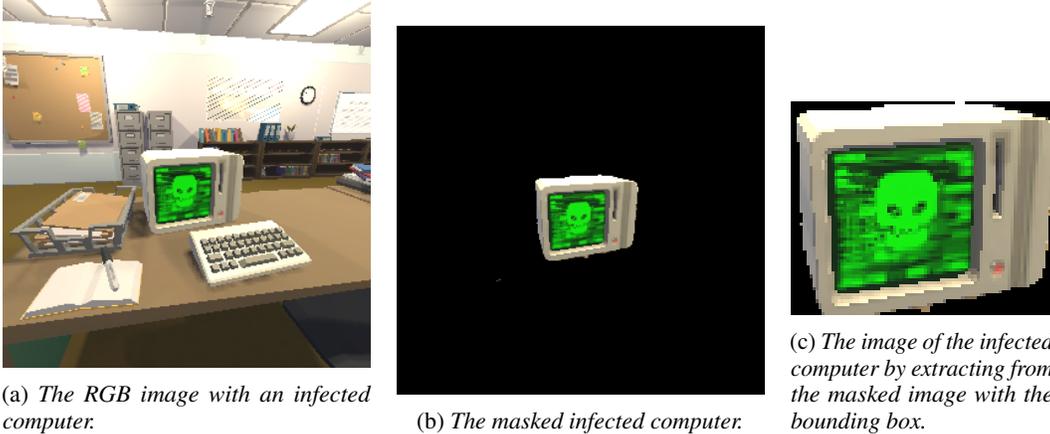


Figure 5: To obtain the input for the CLIP visual encoder, we follow a multi-step process. Firstly, we obtain a masked RGB image by multiplying the original RGB image (a) with a binary instance segmentation mask, resulting in the masked RGB image (b). Next, we calculate the bounding box corresponding to the segmentation mask. Finally, we derive image (c) as the input to the CLIP visual encoder.

We have observed that the CLIP-based approach can also be useful in identifying the color of detected objects, such as distinguishing between the red and blue shelves in the Robotics Lab. Nevertheless, its accuracy is not always reliable when differentiating between the *pink*, *green*, and *black* computers in the Quantum Lab. To improve its performance and ensure its stability, we utilize the determinacy of the Alexa Arena environment and determine the object’s color by calculating the CIEDE [Luo et al., 2001] color difference between pixels within the bounding box and a set of reference RGB values. When determining the computer color in the Quantum Lab, we compare the RGB value of each pixel with three reference colors:

- [200, 20, 130] for *pink*
- [100, 200, 80] for *green*
- [23, 23, 25] for *black*

and assign its class based on the argmin of the distance. Finally, we determine the computer’s color by selecting the color with the most pixels. Note that the referenced RGB values are obtained by eyeballing the most common ones in a specific masked computer image.

While the approaches above achieve satisfactory results when the user specifies the target instance’s color or state, it still fails to reason about the spatial relationships between multiple objects. We discuss this issue in more detail in Section 3.4, where we propose a solution by leveraging the depth images.

2.5 Exploration Engine

When the target object cannot be detected from the current observations, the Exploration Engine takes over and generates an action sequence to search for the object. To accomplish this, we use a variety of strategies, including:

- Execute a *Look Around* action if one has not been performed.
- If the previously interacted receptacle has a door and is now closed, try to open the receptacle to find the target object.
- If the agent just searched the lower fridge, search the freezer for the target object after obtaining permission from the user, and vice versa.
- If the agent has previously detected the target object at a specific pose, try to return to that pose to better locate the target object.



Figure 6: The RGB image observation from the four viewpoints of the robotics lab. Thanks to the design of the Alexa Arena, our agent can easily explore a given room by traversing the eight viewpoints.

- If the object is not PICKUPABLE and cannot be found in the current room, ask the user for the room to find the target object. For example, the *vending machine* can only be found in *Break Room* and *Reception Room*.
- If the agent has previously interacted with the target object, attempt to reuse the history trajectory starting from the most recent viewpoints.
- Traversing the eight viewpoints in the room to search for the target object as shown in Figure 6.
- Ask the user for an anchor object to search around.

By combining these strategies, we can effectively explore the environment and locate the target object, even in cases where it is not immediately visible. However, the current exploration strategies still lack the capability to find the target object inside an openable receptacle that has not yet been interacted with. This requires effective collaboration with the user to identify the correct receptacle.

2.6 Lightweight Dialogue Assigner

In the Alexa Arena simulation environment, certain actions, such as moving to a different room and operating machines, may require several seconds to complete. This can cause frustration for users who must wait for the action to finish before issuing the next instruction. The delay is particularly

problematic for multi-step commands, which can significantly impact the user experience. To address this issue, we have developed a Lightweight Dialogue Assigner module that focuses on generating responses during object interactions. Although the module is easy to implement, we found it to be extremely beneficial as it engages users by explaining the next actions and spicing up the conversations. For example, it adds a lightweight dialogue *Dun dun dun* or *Bang* when trying to break an object with a hammer. As shown in Figure 1, the Lightweight Dialogue Assigner generates the lightweight dialogue *Soda acquisition in process* when trying to pick up the soda can from the desk.

In addition to the lightweight dialogue added to the object interacting actions, we propose copying the user instructions without merely repeating the user’s utterance. Instead, our response is generated based on the predicted actions before translating them into the actual executable action sequence, as explained in Sec. 2.1. Furthermore, the response only takes into account the predicted action, source, and target object names. Consider the example shown in Figure 2, where the predicted action is *Goto*, the source object is *milk*, and the target object is *wall shelf*. Our response would be:

Alright, we’ll take the milk to the red wall.

Note that we omit the object color, room, and anchor object for simplicity, even though the first action we execute is going to the fridge. Similarly, we use the ChatGPT [Open AI, 2023] to generate a set of phrasings for *Alright, we’ll* to add variety to the conversations.

Similarly, when receiving confirmations from the human user, we add a confirmation response as a lightweight dialogue. Specifically, we use Speechcon¹ words selected from the list below to increase expressiveness and make the conversation more enjoyable

gotcha, eggcellent, hear hear, righto, okey dokey, roger, got it.

3 Miscellaneous

In this section, we discuss other techniques we employed to enhance the effectiveness of our agent and improve the user experience, such as adjusting the dialogue response and leveraging depth information to locate the target object accurately.

3.1 Adjusting the Dialogue Excitement

While the standard Alexa voice effectively communicates with users, it may not be expressive enough to engage them fully. To address this issue, we leverage the Alexa Speech Synthesis Markup Language (SSML)² to adjust the excitement level of our robot’s responses. By increasing the excitement magnitude, our bot can convey its eagerness to assist the human user effectively. Moreover, we incorporate interjections to create a more engaging and joyful gameplay experience.

3.2 Find another instance of the same vision class

Sometimes, our agent may move to an object instance that the user does not intend to interact with, and the user may request to search for another instance of the same object type. To address this scenario, we propose recording the agent’s pose after arriving at a specific object, such as a computer in the Main Office. When determining whether a candidate instance is valid, we calculate its center coordinates with respect to the world frame and compare them with the object coordinates of the same class that have already been visited. This way, we can avoid revisiting the previously searched object. We note that this feature is particularly useful when searching for the infected computer in the Main Office.

3.3 Accurately locate the colored Computer in the Quantum Lab

In the Alexa Arena environment, the Quantum Lab has a large number of unused lab terminals, which can create distractions when searching for the computer controlling the gravity pad, time portal, or

¹<https://developer.amazon.com/en-US/docs/alexa/custom-skills/speechcon-reference-interjections-english-us.html>

²<https://developer.amazon.com/en-US/docs/alexa/custom-skills/speech-synthesis-markup-language-ssml-reference.html>



(a) *Make a bowl of cereal with milk.*



(b) *Open the cabinet above the time machine.*

Figure 7: (a) Our model currently struggles to understand abstract user commands. (b) There are three cabinets in the images. To identify the target, our model must reason about the relationships between different objects.

embiggenator. To address this issue, we observe that these three target computers are all located on the tables while the distractors are stacked on the floor. Leveraging the depth information from the Alexa Arena engine, we generate a mask that only includes the space above the tables. This allows us to effectively eliminate all the distractor computers and locate the target computer accurately.

3.4 Accurately Identifying Targets in the Presence of Multiple Instances from the Same Class

The visual component of an EAI agent serves as its eye, enabling it to perceive and interact with the environment. Although state-of-the-art instance segmentation techniques [He et al., 2017, Cheng et al., 2021] can extract object masks from RGB observations, accurately distinguishing between multiple instances of the same object class poses a challenge. In our current framework, we address this issue by utilizing the CLIP model [Radford et al., 2021] to select the best instance corresponding to the user’s utterance. However, this method is limited in its ability to reason about the relationships between different objects. As a result, our agent may struggle to identify the correct object when the user’s instructions involve the relative positions of other objects, as illustrated in Figure 7b, where the user might request the agent to *Open the cabinet above the time machine.*

To address the challenge mentioned above, we propose leveraging depth information to compute the 3D coordinates of the candidate object and the anchor object with respect to the world frame. As described in Section 2.1, we can extract the target object name and the anchor object by parsing the user’s utterance in a structured manner. In Figure 7b, for instance, the target object is the *cabinet*, and the anchor object is the *time machine*. Next, we eliminate half of the candidate *cabinets* based on the preposition used in the user’s sentence. In this example, since the user is looking for the cabinet *above*, we exclude all candidates that are below the countertop. This can be easily accomplished because all cabinet instances that are below the countertop are assigned to the *Counter Base* vision class. In contrast, the cabinets above the countertop are assigned to the *Cabinet* vision class. Finally, we select the candidate closest to the *time machine* by calculating the Euclidean distance. Note that we can follow a similar procedure to find the cabinet *below* the time machine.

4 Future Work: Understand Abstract User Commands

While the current version of our GauchoAI has obtained a satisfying performance on the 14 tasks released on the Alexa Arena Runtime system, there are various perspectives on which we can improve our agents. Specifically, enabling our bot to understand abstract human commands will be critical to make it a user-friendly robot assistant.

As mentioned in the previous section, the Action Prediction Engine that generates action sequences from user utterances is presently based on rules. While it can tackle under-specified user utterances to some extent, such as *Pick up the apple from the fridge* that omits *Open the fridge*, thanks to our powerful pre-processing engine, it struggles to comprehend more abstract user utterances that convey multiple sub-goals for the agent. For example, *Make a bowl of cereal with milk*. To address this issue, we propose fine-tuning a large language model (LM) to translate abstract user utterances before inputting them into our existing Action Prediction Engine.

We aim for the output of the LM to be compatible with our current Action Prediction Engine. Our approach is distinct from autoregressively training a conventional action sequence predictor that must predict the exact next action for execution. Both the input and output are from the natural sentence space, allowing us to fine-tune an off-the-shelf sequence-to-sequence translation model, such as BART [Lewis et al., 2019] and Flan-T5 [Chung et al., 2022], to accomplish this task. Consider the user command *Make a bowl of cereal with milk* again. We expect the fine-tuned language model to rephrase the original utterance into *Pour the cereal into a bowl and pour the milk into a bowl*, which can be handled within our current framework.

5 Supplementary

We will comply with Amazon’s guidelines and release the source code accordingly. Interested readers can refer to our code for further implementation details, including our dialogue strategies.

References

- Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.
- Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR, 2022.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. *Advances in Neural Information Processing Systems*, 34:17864–17875, 2021.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.
- Qiaozi Gao, Govind Thattai, Xiaofeng Gao, Suhaila Shakhiah, 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.
- Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- Zhiwei Jia, Kaixiang Lin, Yizhou Zhao, Qiaozi Gao, Govind Thattai, and Gaurav S Sukhatme. Learning to act with affordance-aware multimodal neural slam. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5877–5884. IEEE, 2022.
- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

- M Ronnier Luo, Guihua Cui, and Bryan Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 26(5):340–350, 2001.
- So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*, 2021.
- Open AI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- 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.
- Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic transformer for vision-and-language navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15942–15952, 2021.
- 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.
- Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.
- Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. *arXiv preprint arXiv:2110.01517*, 2021.