Learning Multimodal Interaction Manager for Assistive Robots from Human-Human Data

Anonymous Author

***Abstract*— This paper describes a Reinforcement Learning (RL) framework for developing assistive robots capable of multimodal interaction. The framework critically depends on a neural network-based human user simulator trained on the existing ELDERLY-AT-HOME corpus, accommodating multiple modalities such as language, pointing gestures, and haptic-ostensive actions. The simulator provides a multimodal interactive environment for training the Reinforcement Learning (RL) agents in collaborative tasks involving various modes of communication. In contrast to conventional dialog systems, our agent is trained using a simulator developed with human data and capable of handling multiple modalities, including language and physical actions. The paper also presents a novel multimodal data augmentation approach, which addresses the challenge of using a dataset that is small due to the expensive and time-consuming nature of collecting human demonstrations. Overall, the study highlights the potential for using RL and multimodal user simulators in developing and improving assistive robots.**

I. Introduction

Assistive robots have the potential to transform the lives of older adults, their caregivers, and people with disabilities. They can be designed to help with activities of daily living (ADLs) such as cooking and cleaning. In assistive robots, the traditional sense-plan-act architecture can be realized through three core components: the Perception Module, responsible for comprehending sensory inputs from multiple modalities; the Execution Module, carrying out the desired actions; and the Interaction Manager, which is responsible for the robot's decision-making and thus perhaps most critical for the human-robot interaction (see Figure 1).

The main focus of this work is on automatically learning the Interaction Manager. The main challenge to doing so is the multimodal nature of a typical interaction that includes high-level modalities such as language, and low-level modalities such as haptic signals, and the difficulty of obtaining training data with such interactions. Capturing realistic interactions is time-consuming. Further, we lack sensors that can adequately record haptic signals, which are often the dominant modality during the shared task, in unrestricted settings.

In [1], the focus is on the "Find" task, an interaction scenario in which two partners work together to find an object in the environment. The proposed framework for the Interaction Manager for the Find task is Hierarchical Bipartite Action-Transition Networks (HBATNs). They model both agents simultaneously to maintain the state of a task-driven multimodal interaction and plan subsequent robot moves. This framework was developed using a subcorpus of the ELDERLY-AT-HOME corpus [2], a publicly available corpus that captures real-world interactions between nursing

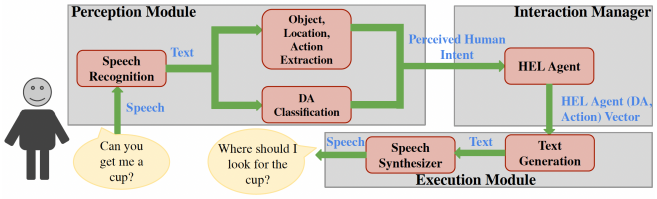


Fig. 1: The *Sense-Plan-Act* cycle in an assistive robot

students (HEL) and elderly individuals (ELD) engaged in activities of daily living. Unfortunately, HBATNs are constructed by hand and thus difficult to scale up and adapt to new tasks.

Our goal is to substitute manually crafted robot policies with policies generated automatically through reinforcement learning (RL), to improve assistive robot capabilities. Training RL agents for multimodal human-robot interaction (HRI) is challenging because it requires interactive environments that can provide RL agents with meaningful rewards. To overcome this challenge, we're introducing a novel neural network-based user simulator inspired by Behavioral Cloning [3], [4]. Our user simulator is unique because it accommodates multiple modalities such as language, pointing gestures, and haptic-ostensive (H-O) actions [2]. This is an important advance in developing domestic assistive robots as there is currently no simulator that can provide a multimodal interactive environment for RL training.

Developing and training such a user simulator is challenging due to the limited amount of data and its sparsity. Another significant contribution of our work is a novel multimodal data augmentation approach that addresses this challenge. Our data augmentation approach effectively takes care of all modalities involved in the data, reducing the need for expensive and time-consuming human demonstrations.

Our proposed user simulator also simplifies the often challenging process of defining and fine-tuning reward functions for multimodal HRI tasks. The simulator allows for greater ease in creating reward functions that can be readily applied to a variety of other tasks, enhancing their generalizability.

Overall, our study highlights the potential for using RL and multimodal user simulators in developing and improving domestic assistive robots.

II. Related Work

Machine learning has become an indispensable tool in robotics, enabling intelligent agents to interact with their environment effectively. RL stands out as a promising approach for learning policies from sequential human-robot

interaction data. Since the seminal work on Q-learning in the '80s by Watkins and Dayan [5], RL algorithms have advanced significantly, incorporating deep learning techniques such as Deep Q-Learning [6], Actor-Critic [7], and Trust Region Policy Optimization [8]. Notably, RL agents have demonstrated superhuman performance in diverse domains, as exemplified by their success in Atari games [7].

Inverse reinforcement learning (Inverse-RL) has gained prominence, particularly in scenarios where data is limited, and human guidance can enhance the learning process. Inverse-RL models seek to understand human preferences and behaviors by observing their actions. For instance, [9] introduced a preference-inference Inverse-RL model for assistive robots, which learned user preferences during task execution. In this work, we delve into the challenge of training an intelligent assistive robot agent and propose a Deep Q-Learning (DQL) framework enhanced with a DAGGER warm-up [10]. This enhancement facilitates the efficient tuning of the RL reward function.

Simulated users or interactive environments are indispensable components in RL, serving as a crucial part of the training feedback loop. In the context of human-robot interaction, these simulated users should not only be reliable but also capable of generating a diverse range of actions to facilitate RL exploration. Thomaz and Breazeal [11] investigated the compatibility of human-given rewards with traditional RL reward signals.

In recent years, there has been a growing body of work focusing on RL frameworks for training interactive dialogue systems designed to assist users [12]–[14]. For instance, Li et al. [14] proposed a Deep Q-learning network architecture for a dialogue system aimed at assisting humans in accessing information and accomplishing tasks. Notably, this dialogue agent was trained with a language generation machine as the user simulator. A key differentiator between this research and ours is that we leverage actual human data to develop our user simulator. Furthermore, while dialogue systems are typically unimodal, focusing solely on language, our work tackles the complexity of a multi-modal system encompassing gestures, physical actions, and dialogue.

Another relevant study is that of Park et al. [15], which employed Q-learning to learn a personalized policy based on children's facial reactions and pose data in a classroom setting. However, this work solely relied on visual data captured by cameras and did not encompass multi-modal interactions. Our research takes a distinctive approach by incorporating various modalities, such as gestures, physical actions, and dialogue, into the training process based on actual human data.

Our work builds upon the HBATNs framework introduced in [1], [16]. This framework facilitates human-robot interaction by allowing the agent to manipulate objects in the environment and engage with users through gestures, H-O actions [2], and speech. The HBATNs model, roots in data-driven techniques and informed by the human-human interaction corpus [2], is particularly tailored to the collaborative "Find" task. While HBATNs provide an interpretable decision-making process, they are manually constructed and lack scalability. Importantly, policy extraction is a labor-intensive manual process. A primary contribution of this paper is the automation of this policy extraction process through RL.

Simulators play a vital role in advancing robot autonomy through machine learning. In the context of HRI, developing simulators that accurately model complex human behavior is a significant challenge. Some approaches model human movement through equations of motion [17], [18], while others assume human agents have nearly identical reward functions as robots [19]. These assumptions simplify simulator design but may not fully capture human behavior. In contrast, our approach models human agents based on real-world data, relaxing some of these assumptions and allowing for a more nuanced understanding of human behavior and intentions.

Recent advancements in intelligent dialogue systems have shown promise in modeling the dialogue aspect of human agents [14], [20], [21]. While some of these systems focus on end-to-end training, they typically operate within single-modal domains, primarily language. In contrast, our work extends to a multi-modal setting that encompasses gestures, physical actions, and dialogue based on actual human data. Our approach represents a valuable contribution to the field of HRI, which currently lacks comprehensive studies considering multi-modal interactions in training intelligent agents based on real human data.

III. User Simulator Framework[1]

A detailed description of the background information regarding this section is provided in Appendix A.

A. Feature Extraction

In this section, we introduce an end-to-end model to predict the state of the world that includes the ELD state as well as the ELD's next move.

Our model is implicitly supposed to act as the ELD. However, our analysis of the data revealed that HEL could take consecutive moves. In order to address those cases where the HEL is taking more than one move, the model should determine whether or not the ELD will take an action in the subsequent move. In summary, the model's objective is twofold: (1) To predict the ELD's next action and whether it will occur. The ELD's move is determined by the ELD's Dialogue Act (DA) and the ELD's action. (2) To determine the state of the world. The state of the world is determined by the ELD's belief of the HEL's knowledge of object type, location, and object.

In order to effectively train our end-to-end model, we must first determine what information can aid the model to decide the ELD's state and its action. The following items summarize the important points that should be taken into account when selecting the features and how we have featurized and annotated our collected human-human data.

[1]This section has been previously presented at a conference.

Consecutive Moves and Previous Actor: To determine whether the HEL will take consecutive moves, we need to know the previous actor. This information can be represented as a two-element vector. If the trial has not started and the HEL initiates it, both elements are set to 0. If the HEL isn't going to take consecutive moves, the previous actor is the ELD, and the first element of the vector is set to 1. Otherwise, the second element is set to 1.

Object Type and Location Utterances: It is important to know whether object types or locations have been mentioned. These can be represented as binary features.

ELD's Previous State: The model also needs to have information on the ELD's previous state. We propose the following representation for it: (1) ELD's belief of HEL's knowledge of O_T, which can be one of three values (ELD believes HEL does not know the target $O_T \rightarrow 0$, ELD believes HEL knows the target $O_T \rightarrow 1$, or ELD believes HEL is thinking of a different $O_T \rightarrow 2$), (2) ELD's belief of HEL's knowledge of L, and (3) ELD's belief of HEL's knowledge of O. We extracted all the meaningful possible combinations of these three parameters which would give us 13 distinct combinations as follows: (1) state $(0, 0, 0)$, (2) state $(0, 1, 0)$, (3) state $(0, 2, 0)$, (4) state $(1, 0, 0)$, (5) state $(1, 0, 1)$, (6) state $(1, 0, 2)$, (7) state $(1, 1, 0)$, (8) state $(1, 1, 1)$, (9) state $(1, 1, 2)$, (10) state $(1, 2, 0)$, (11) state $(2, 0, 0)$, (12) state $(2, 1, 0)$, (13) state $(2, 2, 0)$. We use one-hot encoding, a one-hot encoded vector of size thirteen, for representing these states. The elaborated description for obtaining these states is provided in section III-F.

Pointing Gesture by HEL: We consider whether the HEL has performed a pointing gesture and if the target is an object or a location. This categorical feature is represented as a five-element vector. The first two elements indicate whether a pointing gesture has occurred (0 if not). The first element is 1 if the HEL points to a location, and the second element is 1 if the HEL points to an object. The last three elements determine whether the location or object pointed to by the HEL matches the ELD's location/object. If the third element is 1, it indicates a match, while the fourth element represents a mismatch. The fifth element being 1 means that the HEL points to the correct object type but not the specific object. For example, imagine that the ELD asks for a small bowl and the HEL points to a large bowl in response; the vector $(0, 1, 0, 0, 1)$ is the indication of the HEL's pointing gesture.

H-O Actions: Likewise, in the case of H-O actions, we must ascertain whether the action has taken place and whether it was directed toward an object or a location. In addition, we also require the type of H-O (opening or closing a location, touching a location or an object, taking out an object, holding an object). This is a categorical feature, a vector of size ten. The first five elements are interpreted exactly the same as the pointing gesture vector. The second half of the H-O action vector determines which of the five different H-O actions has been performed by one-hot encoding those action types.

Current HEL's Action and Utterance: We also add the current HEL's action and the current HEL's utterance (DA tag), which are both categorical features represented by vectors of size nine and fourteen respectively. The HEL action classes are categorized as follows: (1) No action, (2) Request O_T, (3) Request L, (4) Verify O_T, (5) Verify L, (6) Verify O, (7) Acknowledge, (8) Yes, (9) No. The DA classes are categorized as follows: (1) No utterance, (2) Instruct, (3) Acknowledge, (4) Query-w, (5) Query-yn, (6) Reply-w, (7) Reply-y, (8) Reply-n, (9) Check, (10) Explain, (11) Align, (12) State-y, (13) State-n, (14) State. These were the DAs chosen to annotate the "ELDERLY-AT-HOME" corpus [2].

Previous ELD's Action and Utterance: In addition to the current HEL info, our model relies on the information from the previous actions from the ELD. Here we use the ELD's action and ELD's DA which are both categorical features represented by vectors of size seven and fourteen respectively. The ELD's action classes are categorized as follows: (1) No action, (2) Give O_T, (3) Give L, (4) Give O_T, L, (5) Acknowledge, (6) Yes, (7) No. The ELD's DA classes are the same as HEL's.

B. Data Annotation

The *Find* task data in the ELDERLY-AT-HOME corpus [2] was previously transcribed and annotated for DAs, pointing gestures, and H-O actions. As explained in section III-A, we need to provide the network with additional features for training our user simulator. We performed additional annotations for ELD beliefs of HEL's knowledge of O_T, L, and O, and ELD and HEL actions based on the classes introduced for each feature in the previous section.

ELD's perceptions of HEL's knowledge of O_T, L, or O were objectively determined based on the heuristic that ELD updates its beliefs whenever HEL demonstrates knowledge or lack thereof of these entities. For instance, ELD assumes that HEL is unaware of the O_T or L until HEL acknowledges ELD's description of it or takes action to confirm it. If HEL selects the wrong O_T or L when ELD has specified a particular one, ELD assumes that HEL is thinking of a different O_T or L. For example, if ELD points to a small bowl and says "Get me that bowl," but HEL asks "That bowl?" while pointing to a large bowl, ELD believes that HEL is thinking of a different bowl.

Two annotators labeled the actions of ELD and HEL. As the labeling of action did not follow a strict guideline and was, therefore, more open to interpretation, the inter-annotator agreement was measured for both types of actions using Cohen's kappa. To test the reliability of the annotation, 40 random ELD actions and 40 random HEL actions were chosen from the data set, and both annotators labeled them independently before labeling the remaining actions. The results showed a high level of agreement between the annotators for both ELD actions ($\kappa = 1.0$) and HEL actions ($\kappa = 0.81$), indicating that the action labels are reliable.

C. Multimodal Data Augmentation

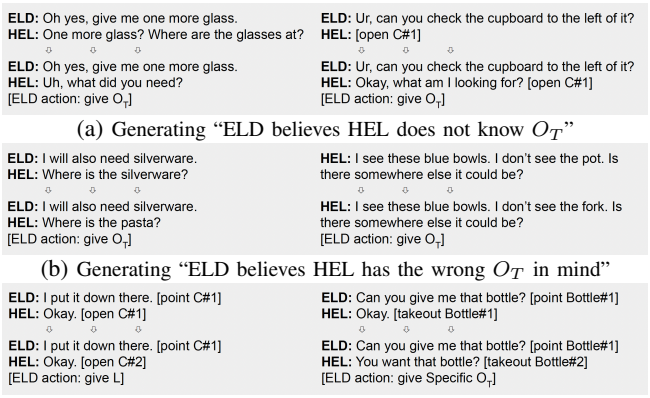
The data collected during the *Find* task is suitable for building a strong basis to train the user simulator; however, there are no instances or few instances in which ELD

believed HEL had the wrong O_T or L in mind or did not know the O_T. This lack of variation is not surprising, as the interactions between the two humans were relatively straightforward. However, since we propose this user simulator as the main component of the interactive environment for the RL training, and as we expect the HEL agent, trained with reinforcement learning, to make mistakes, we need to augment the data to include examples of missing or infrequent states.

To increase the number of instances in which ELD believes HEL does not know the O_T, states (0,0,0) and (0,1,0), we sample instances in which ELD believes HEL knows the O_T and replace HEL's utterance and action with an example of requesting the O_T (see Fig. 2a). The ELD then gives instructions on O_T again.

To increase the number of instances in which ELD believes HEL has the wrong O_T in mind, states (2, 0, 0) and (2, 1, 0), we sample instances in which HEL mentions the target O_T in their utterance and replace it with an incorrect one (see Fig. 2b). The ELD then gives instructions on O_T again.

We also increase the number of instances in which ELD believes HEL is not thinking of the same L or O, states (1, 2, 0) and (1, 1, 2), by sampling instances in which HEL's utterance or action includes an object or location and replacing it with an incorrect one. In each case, ELD would then give the correct O_T or L again (see Fig. 2c). These synthetic examples will help the user simulator respond appropriately to mistakes made by the HEL agent.



(a) Generating "ELD believes HEL does not know O_T"

(b) Generating "ELD believes HEL has the wrong O_T in mind"

(c) Generating "ELD believes HEL is not thinking of the same L or O"

Fig. 2: Examples of how we augment our data with unseen or infrequent states

Early on for developing a Basic User Simulator (BUS model), we conducted data augmentation exclusively for ELD's output states. However, after carefully examining the data, we observed that only the first nine state combinations mentioned previously are included in the input states whereas all thirteen combinations could be seen in ELD's output states even if they are rare. That being said, state combinations 10 to 13 are meaningful and highly likely to happen during an interaction between the user simulator and the HEL robot agent. Not seeing all possible meaningful inputs during the training, causes the model to be too specific and not able to handle all possible situations it may encounter. It also makes the final accuracy evaluations not accurate because the model has been trained, validated, and tested on inputs that do not offer enough variation. Thus, we augment the data points in such a way that all thirteen state combinations are covered in inputs so that we make our model more accurate and flexible.

To synthesize the input state where in the previous move ELD believed HEL had the wrong O in mind, state (1,1,2), we randomly choose some instances where the input state is (1,1,1), i.e. in the previous move ELD believed HEL had the correct O in mind, and change the ELD's previous move accordingly. For that, ELD would inform the HEL about the object again. So we replace the previous ELD's DA and action with "Instruct" and "Give Specific O_T" respectively. We should point out that later on, we combined actions "Give Specific O_T" and "Give O_T" into one single class as during the interaction with the HEL, these two actions convey the same message and only the difference in ELD's state matters when ELD announces either the object type or the specific object during the interaction with HEL.

To synthesize the input states where in the previous move ELD believed HEL had the information about O_T and L but had wrong O_T and/or wrong L in mind, states (1,2,0), (2,1,0), (2,2,0), we randomly sample our data points where in the previous move ELD believed HEL had the right O_T and L in mind and change the ELD's previous move accordingly. For state (1,2,0), ELD would inform the HEL about the location again, so we replace the previous ELD's DA and action with "Instruct" and "Give L" respectively. Similarly, for the state (2,1,0), ELD would inform the HEL about the object type again, so we replace the previous ELD's DA and action with "Instruct" and "Give O_T" respectively. Lastly, for the state (2,2,0), ELD would inform the HEL about both object type and location again, so we replace the previous ELD's DA and action with "Instruct" and "Give O_T, L" respectively.

To synthesize the input states where in the previous move ELD believed HEL had the wrong O_T in mind and didn't have any other information about L and O, state (2,0,0), we take random samples where in the previous move ELD believed HEL had the correct O_T in mind and change the previous ELD's DA and action to "Instruct" and "Give O_T" respectively.

With this data augmentation scheme, we increase our data points from 693 to 1932.

D. Model Architecture and Training

Our model is a neural network consisting of three fully connected (dense) layers, and a dropout layer (ratio=0.2) to prevent overfitting and improve the ability of the model to generalize better. We utilized the Cross-Entropy loss function and Adam optimizer during training. The training process lasted for a maximum of 100 epochs, but we also evaluated the model's performance on the validation set while training to allow for early stopping. The inputs to the neural network were the features described earlier, while the outputs were the ELD's next state, dialogue act, and action, which were

manually annotated in the data. We implemented the model using the PyTorch library [22].

E. Model Evaluation on Data

The model was trained on 80% of data (a total of 1548 data points). About 10% of the data (a total of 183 data points) was used for validation purposes during the training to early stop the training before over-fitting happens. The rest of the data (a total number of 201 data points) was used for evaluating the performance of the fully trained model. To better capture the great performance of the GUS model, we compare it to the BUS model.

In addition to overall accuracy, we evaluated the model on the classification accuracy of each individual output of the model; i.e. the classification accuracy for (1) the predicted ELD's action; (2) the predicted ELD's DA; (3) the predicted ELD's state, the ELD's belief of the HEL's (O_T, L, O).

Here, we report the accuracy results of BUS and GUS models tested on the original and augmented test data sets. The results are summarized in Table I.

	Overall Acc.	Action Acc.	DA Acc.	State Acc.
BUS, Org. Test Data	46.27%	52.22%	53.73%	83.58%
BUS, Aug. Test Data	45.83%	56.77%	51.04%	68.75%
GUS, Org. Test Data	66.67%	81.82%	72.73	93.94%
GUS, Aug. Test Data	70.85%	77.89%	75.38	89.44%

TABLE I: Classification Accuracy of BUS and GUS Models

The results in Table I show a great improvement from the BUS model to the GUS model. This significant improvement results from the changes we made to our feature extraction, data augmentation, and the model architecture itself that can be summarized as (1) changing the ELD's state representation as explained in detail in section III-A, (2) Giving information about ELD's previous DA and action as input to the model, (3) Augmenting ELD's input states as explained in detail in section III-C, (4) Adding layers to the network architecture and removing nonlinear ReLU activation function from the network, (5) Not executing sample re-weighting since the class imbalance issue is already resolved by proper data augmentation and we don't want to overdo re-weighting because of the limited amount of data we have. Considering the small amount of data, re-weighting actually affects the performance of our model adversely.

To further analyze the evaluation results reported in Table I, we investigate the corresponding confusion matrices. Before moving forward with the confusion matrices, one should remember that as explained in section III-A, our trained model is supposed to also decide whether or not the ELD will take a move. If that's the case when the HEL is taking two consecutive moves, the model would output *0* and *None* as the predicted ELD's action and DA respectively.

The DA confusion matrix in Table B.1 brought in Appendix B shows that the main confusion is due to wrongly classified "Instruct" DAs as "Reply-n". One justification for this confusion is the limited number of data, and on top of that is the imbalanced classes. However, going over the annotated data, we see that most of the time when the ELD responds to a verification question or O_T/L query ("Query-yn/Cehck" and "Query-w" DAs respectively), s/he carries on with giving further instructions. That means in many cases where ELD's utterances are labeled as "Reply-n", they could also be interpreted as "Instruct" and eventually convey the same intent to the HEL. For instance, the HEL verifies the O_T by asking the ELD "Did you say a pot?", and the ELD replies with "No, get me a bowl.". The ELD's DA could be labeled as either "Reply-n" or "Instruct", but what matters here is that the HEL receives the same intent from ELD.

Going through the confusion matrix in Table B.1 in Appendix B, we observe that the rest of the DA classes are either classified very well or confused with classes that don't influence the overall outcome of the network. For instance, the DA "Acknowledge" is correctly classified in 66.67% of the cases and has been classified as "State-y" in 33.33% of the cases. However, this misclassification doesn't affect the message that HEL receives.

Analysis of the action confusion (the results presented in Table B.2 in Appendix B) shows that 22.06% of the "Give O_T" samples are wrongly classified as "Yes". This is again due to the limited number of data points available as well as imbalanced classes. However, analogous to our explanation above, this misclassification could be because of those cases where the ELD responds to a verification question and continues by giving instructions. For example, the HEL verifies the L by asking the ELD "Did you say that cabinet?", and the ELD replies with "Yes, get me the silverware.". The ELD's action could be labeled as either "Yes" or "Give O_T". In either case, the HEL would receive the same message.

We also observe that 33.33% of the "Acknowledge" classes are misclassified as "Yes" actions. Again, because these two actions are inherently very similar, this misclassification doesn't affect the message that HEL receives.

In summary, many wrong DA and action classifications are due to the fact that distinguishing DA classes like "Instruct", "Reply-y", "Reply-n", "State-y", "State-n", and action classes such as "Give O_T", "Give L", "Acknowledge", "Yes", and "No" would be very difficult. This happens because in our available data, most of the cases where the ELD responds to a "Query-yn" question, start with saying yes, no, acknowledging, and then guiding the HEL towards a location and/or giving information about the object.

F. Model Evaluation on HBATN

To further investigate how realistic our model performance is, we compare its performance to the previously developed HBATN model. It is important whether or not the GUS model acts similarly to HBATNs because HBATNs are carefully hand-crafted by human annotators who based their insights in the data. For the purpose of this comparison, we would need to have the response of our trained model to variant inputs.

To generate different meaningful inputs for the GUS model, we employ an automatic approach. Before moving

to explain the approach itself, we need to lay some ground rules as follows:

(1) For the ELD's state representation, ELD's belief of HEL's knowledge of O_T cannot change from 0 to 1 or 2 before ELD utters the O_T. (2) ELD's belief of HEL's knowledge of L cannot change from 0 to 1 or 2 before ELD utters the L. (3) ELD's belief of HEL's knowledge of O cannot change from 0 to 1 or 2 before ELD's belief of HEL's knowledge of O_T turns to 1. (4) ELD's belief of HEL's knowledge of O can change from 0 to 1 or 2 before ELD's belief of HEL's knowledge of L turns to 1. (5) HEL cannot verify O_T and L before ELD announces them. (6) HEL only performs pointing and H-O actions for $O_T/L/O$ verifications.

The ground rules associated with the ELD's state representation would give us thirteen distinct meaningful combinations which we explained in detail in section III-A. After combining those states with other inputs, by applying the rest of the ground rules to all combinations, we generate all meaningful inputs automatically.

Subsequently, we put our GUS model in different states that previously had ben extracted from the HBATNs [16] by applying different inputs to the model and finally we compare the GUS model outputs to those of HBATNs.

The results of comparing our GUS model to the HBATNs are summarized in Tables II, III, IV. In the (a, b, c) tuple which represents the input or the output, c stands for the HEL DA (input) or ELD DA (output), a and b also determine whether or not the O_T and L have been uttered respectively. Each input/output also includes ELD's previous state, "pointing/H-O" actions, ELD's previous DA and action, and HEL's DA and action features. We omitted these parameters in our table representations for simplicity. However, we carefully mapped our automatically generated inputs to different HBATNs states for these comparisons.

Our comparisons illustrate that the GUS outputs greatly match the HBATNs. There are only a few minor differences that don't affect the interaction between the ELD and the HEL and their intents. For example, our GUS model confuses "Instruct" and "Reply-w" DAs in some cases such as in Table II, the primitive subtask "Establish(O_T)" where it only outputs "Reply-w" for all "Instruct/Reply-w"-labeled outputs. For the HEL it doesn't matter if the utterance is labeled as either one because both DAs transfer the same message.

In some other instances, such as in Table III, the primitive subtask "Specify(O_T)", our GUS model outputs "Instruct/Reply-y/Reply-n" DAs for "Instruct"-labeled outputs. This again is not a fatal error because the action the GUS model outputs as the ELD's action would provide instructions about the O_T or L. In Table IV, for the primitive subtask "Finish(L)", the GUS model confuses "Acknowledge" DA with "Reply-y/State-y". This is also negligible due to the similar inherent that these DAs carry. Nevertheless, this is the final action in the interaction and wouldn't affect the interaction at all.

We should also point out that our GUS model differs from the HBATNs in some cases where the ELD in HBATNs utters or doesn't utter the O_T or L. In a few cases, the GUS model utters O_T or L when the ELD in HBATNs doesn't or vice versa. This minor error is due to applying machine learning approaches to build an artificial intelligence agent as our user simulator. Although these minor errors could make the interaction longer or unsuccessful in very few cases, our GUS model would still be superior to the hand-crafted framework. The next section demonstrates that our user simulator can be successfully used in practice for the RL training of assistive robots.

	Input	HBATN Output	GUS Output
Establish(O_T)	$(0, 0, Inst)/(0, 0, Qw)$	$(1, *, Inst)/(1, *, Rw)$	$(1, 0, Rw)$
Verify(O_T)	$(1, 0, Chk)/(1, 0, Qyn)$	$(*, 0, Ry)/(*, 0, Rn)/(1, 0, Inst)/(1, 0, Rw)$	$(1, 0, Ry)/(1, 1, Rn)/(1, 0, Inst)/(1, *, Rw)$
Specify(O_T)	$(*, 0, Qw)$	$(*, 0, Inst)/(*, 0, Rw)$	$(*, 0, Rw)$

TABLE II: GUS performance evaluations for Det(O_T) subtask. $(*)$ represents 0 or 1.

	Input	HBATN Output	GUS Output
Establish(L)	$(*, 0, Inst)/(*, 0, Qw)$	$(*, *, Inst)/(*, *, Rw)$	$(*, *, Inst)/(*, *, Rw)/(*, *, Ry)/(*, *, Rn)$
Verify(L)	$(0, 0, -)/(0, *, Chk)/$ $(0, *, Qyn)$	$(0, *, Ry)/(0, *, Rn)/(*, *, -)/$ $(*, *, Rw)/(*, *, Inst)$	$(0, *, Sty)/(0, *, Ry)/(0, *, Rn)$
Specify(L)	$(*, 1, Qw)$	$(*, *, Inst)/(*, *, Rw)$	$(0, *, Inst)/(0, *, Rn)/(*, *, -)$

TABLE III: GUS performance evaluations for Det(L) subtask. $(*)$ represents 0 or 1.

IV. Reinforcement Learning Framework

In this RL problem, we employ a Deep-Q-Network [23] model to act as our assistive robot (HEL) agent by having it interact with the interaction environment containing the BUS model. To enhance the training, we warm up the agent before starting the training loop using DAGGER, an Imitation Learning (IL) algorithm [10]. The warm-up phase happens before initiating the RL training cycles. This step is crucial due to the fact that learning the appropriate agent behaviors with a simple reward function is extremely difficult and training such an agent from scratch is tedious.

The flow of the interactions between the HEL agent and the user simulator while training the HEL agent is illustrated in Fig. 3. The state of the HEL agent consists of three variables: the state of HEL's O_T, HEL's L, and HEL's O. The possible values for each variable are 0, 1, or 2. The variable is encoded with 0 when it has not been determined yet, with 1 when it matches the user simulator's belief about it, and with 2 when there is a mismatch between the HEL value and user simulator's belief.

As depicted in Fig. 3, the HEL agent takes the user simulator's action and the previous HEL's state as input and outputs a DA tag and an action vector encoding the HEL's utterance and physical action, respectively.

A. Model Architecture

The HEL agent network includes two fully connected layers followed by a dropout layer (ratio=0.1). Then the output of the dropout layer is fed to the output layer followed by ReLU activation and gives a vector encoding HEL's (DA, action) output pair. For implementation, we used the Pythorch library [22].

B. DAGGER Warm-up

First, we have our HEL agent interact with the user simulator and run the DAGGER algorithm as a warm-up stage. We don't let the agent get fully trained, we only use it to obtain a good initial guess for the subsequent RL training.

	Input	HBATN Output	GUS Output
Specify(O_T)	$(*, 0, Qw)$	$(*, 0, Inst)/(*, 0, Rw)$	$(*, *, Inst)/(*, *, Rw)/(*, 0, Ry)/(*, *, Rn)$
Verify(O)	$(*, 0, -)/(*, 0, Chk)/$ $(*, 0, Qyn)/(*, *, St)$	$(*, 0, Ry)/(*, 0, Rn)/(*, 0, Rw)/(*, 0, Inst)$	$(*, 0, Ry)/(*, 0, Rn)/(*, *, Inst)$
Finish(L)	$(*, *, Sty)/(*, *, St)/$ $(*, *, Stn)$	$(0, 0, Ack)$	$(0, 0, Ry)/(0, 0, Sty)$

TABLE IV: GUS performance evaluations for Det(O) subtask. ($*$) represents 0 or 1.

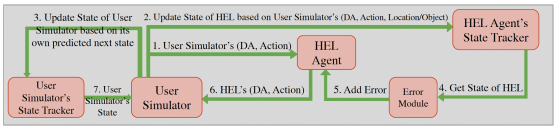


Fig. 3: The interaction between the User Simulator and the HEL agent during RL

Using DAGGER only, without subsequent RL training, results in poorer performance of the HEL agent.

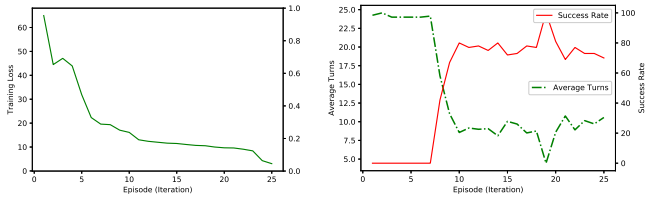
The user simulator initiates the interaction, the state of the HEL agent is updated according to the received input and it picks an action according to its current state and the user simulator's action. An error module is deployed before passing the HEL's state to the HEL agent.

The error module is necessary due to the fact that in the data when $O_T/L/O$ information is given to the HEL by the ELD, most of the time the state corresponding to that changes to 1, i.e. the HEL's understanding of $O_T/L/O$ is the same as what human has uttered. However, we would like to generalize the framework better so that it also covers the states when the HEL's understanding of $O_T/L/O$ does not match the human's utterances. Thus, the error module in 25% of the cases where HEL's understanding of $O_T/L/O$ is 1, changes that to 2.

For training the HEL agent using DAGGER algorithm, we need to run the algorithm for N, (here, $N = 25$) iterations. Here we call each iteration one *Episode* since the interactions between the ELD and the HEL are defined to be episodic. During each episode, one entire interaction consisting of at most M, (here, $M = 25$), turns between the HEL agent and the user simulator takes place. That interaction is successful if the HEL agent finds the object the user simulator requested before reaching turn M. Otherwise, the interaction is unsuccessful.

During each episode, the HEL agent executes the current learned policy. Throughout execution, at each turn, the expert's action, which we get from the roll-outs extracted from our ELDERLY-AT-HOME data, is also recorded but not executed. After sufficient data is collected, it is aggregated together with all of the data that was previously collected. Eventually, the cross-entropy algorithm generates a new policy by attempting to optimize performance on the aggregated data. This process of execution of the current policy, correction by the expert, and data aggregation and training is repeated.

Fig. 4 shows training loss, success rate, and average turns during each episode for DAGGER training on the agent for 50 episodes. However, as mentioned before, we do not want a fully DAGGER trained HEL agent. Therefore, the HEL agent's network weights are saved at episode 8 (a good mid-point where the loss is not minimized and the success rate is not maximized yet) to be used later on for training the HEL agent using the Deep-Q-Learning (DQL) algorithm.



(a) Training Loss (b) Success Rate and Average Turns

Fig. 4: DAGGER Algorithm Training Evaluations

C. Deep-Q-Learning

For training our HEL agent in the RL cycle, we need to initialize a *Policy Network* and a *Target Network*. The former network's weights will be optimized for obtaining the optimal policy, and the latter network will be used to track the target Q-values associated with each individual action for input states [23]. The policy we extracted at episode 8 of DAGGER training is used to initialize the weights of the *Policy Network* and the *Target Network* for DQL. The model architectures for the *Policy Network* and the *Target Network* are exactly the same as the model used previously for DAGGER training due to the fact that we simply copy the weights extracted from DAGGER network into these two networks. This makes running the DQL algorithm on our RL framework much more efficient in terms of time and space. Another advantage of warming up the HEL agent by DAGGER training is that by pushing the weights of the policy toward the expert's policy, we don't need to hard code complex human behavior in the *Reward Function*. A simple *Reward Function* combined with DAGGER-half-trained HEL agent makes the DQL algorithm on our HEL agent run much faster.

For our reward function, we considered a small negative reward, $-r$ (here, $r = 1$), for each HEL agent's move. This is to motivate the agent to finish the task sooner than later. If the interaction is unsuccessful and ends by getting to turn M, the transition is penalized by $-2\,r$. If the interaction successfully ends before reaching turn M, (here, $M = 25$), that transition is rewarded as $2\,r$. We also set some ground rules as *Preconditions*. The *Preconditions* are as follows: (1) the HEL agent must not take the action of verifying O_T before O_T is uttered by the user simulator; (2) the HEL agent must not take the action of verifying L before L is uttered by the user simulator; (3) the HEL agent must not take the action of verifying O before both O_T and L are uttered by the user simulator. If any of the *Preconditions* are violated, that action is penalized with a very large negative number $-Z$, (here, $Z = 50$).

The interactions between the HEL agent and the user simulator while running the DQL algorithm also follow the flow in Fig. 3. A tuple of *(state, reward, action, next state)* is stored in the memory at each turn. Every C episodes, (here, $C = 100$), the weights of the *Policy Network* are optimized using the *Mean-Squared-Error* cost, and every $m\,C$ episodes,

the weights of the *Policy Network* are copied into the *Target Network*.

Fig. 5 shows training loss, average reward, success rate, and average turns at each episode for the full DQL algorithm training of the HEL agent. The reason that average turns start to increase and the success rate gradually decreases after about episode 1250 is that the memory of the agent has been filled up, so part of the memory is emptied and starts to get filled with new samples. Catastrophic forgetting has happened at this point (the agent has been over-fitted). We thus use the policy obtained just before this phenomenon occurs.

(a) Training Loss and Average Reward (b) Success Rate and Average Turns

Fig. 5: DQL Algorithm Training Evaluations

V. Experimental Evaluations on RL Agent

To evaluate our framework in the real world, we designed a user interface to have humans interact with the HEL agent obtained through RL. For details on the user interface implementation the reader may refer to Appendix C. In the following section, we explain the details of our evaluation procedure.

A. User Study

We performed a human user study where 9 healthy adults were recruited to interact with our HEL agent. Each subject performed 4 to 5 trials (entire interactions) with the HEL agent adding up to a total of 42 trials.

The hypothetical experiment environment would be a room with a drawer, a shelf, and a cabinet. The user can choose between red, green, and yellow cups and red, green, yellow, and white balls. At the beginning of each trial, objects are randomly scattered in different locations. The user only knows there are aforementioned locations and objects in the room but doesn't know which item is located where.

The subjects were instructed to choose the object of interest at the beginning of the trial, and guide the agent through different locations to find the object of interest.

The data collected in this user study is publicly available [2].

B. Results

Inspired by the study [24], we measured the performance of the system by calculating the accuracy of each component independently as well as the overall quality of the system. For evaluating speech recognition we measured Speech-to-Text (STT) accuracy which is the percentage of words transcribed correctly. We report the average Speech-to-Text confidence score as well.

To evaluate the DA classification, we calculated the accuracy based on the percentage of the DAs classified correctly. For obtaining the ground-truth gold-standard DA labels, two annotators annotated the texts that were previously extracted from the Speech-to-Text component. We chose 40 random sentences and had both annotators independently label them prior to annotating the remaining sentences. We calculated Cohen's kappa and found a moderate level of agreement between the two annotators ($\kappa = 0.6$). Although it's not a high level of agreement, the DA accuracy still is consistent with the test accuracy (67.23%) we got from ELDERLY-AT-HOME data before.

For evaluating the HEL agent itself, i.e. whether or not the pair of (DA, action) outputs of the HEL agent makes sense according to the DA tag and the action vector it gets as inputs, we compare its actions to those of HBATN's (as the ground truth labels) and report the action accuracy. Moreover, we calculate and report the percentage of the HEL's non-eligible actions with respect to the human's original speech. We had two human transcribers listen to the audio data recorded from human subjects. Their transcriptions from audio data matched about 99.2%. A non-eligible action is one where the agent's response does not correspond to the human's previous action. For example, if the human says: "Please get a cup." and the agent responds with: "Did you say inside the cabinet?", this is a non-eligible action. In other words, non-eligible actions are those actions that do not make sense to the human participants in the study based on the speech they uttered. Non-eligible actions could result from speech recognition errors, DA classification errors, and/or wrong actions by the agent itself.

We also report the average length of interactions as well as the success rate over all trials. A successful trial is a trial in which the agent is able to find the human subject's object of interest in less than 15 turns. If the agent reaches the 15th turn without finding the object, that trial is unsuccessful.

Eventually, we asked the participants to rate their experience on a Likert scale of 1-10. A score of 1 meant "significantly worse than expected" and a score of 10 meant "significantly better than expected".

The results of our preliminary user study in Table V show that our robot agent does very well in extracting and performing the optimal policy. Two very important metrics to evaluate how well our overall framework performs are the average success rate of 92.86% and the number of average turns of 8.38. There are a couple of reasons why the success rate doesn't reach 100%. The main reason is that our framework is not solely the robot agent itself. The user interface is implemented on top of the agent network and each component of this user interface introduces errors to the overall framework. These errors make our robot agent take some wrong actions occasionally which leads to taking more turns to complete the task.

The average non-eligible action rate shows that the HEL agent in 30.57% of the cases takes an action that does not

[2]Since the submission is anonymous, the link to data will be provided after paper acceptance.

Avg. Duration	Avg. #Turns	Success Rate	STT Accuracy	Avg. STT Conf.	DA Accuracy	Action Accuracy	Avg. Non-Elig. Act. Rate	Avg. Likert
90s	8.38	92.86%	85.76%	89.61%	69.82%	90.2%	30.57%	7.9

TABLE V: Results of the preliminary evaluation of the HEL agent

make sense to the human at that point. The action accuracy of 90.2% indicates that 9.8% of the actions the robot agent takes are wrong actions. This small rate of wrong actions by the robot agent itself is due to the fact that compared to HBATNs, our agent is trained through machine learning, not rule-based. It covers a broader set of states while getting trained and as a result when it comes to testing it, the process of making decisions is more complicated. About 20% of all actions are non-eligible actions resulting from errors in other components. To investigate this more, we calculate the contribution of each component to those wrong actions.

Table VI summarizes the contribution of each component to those wrong actions taken by the robot agent. 46.73% of non-eligible actions (corresponding to about 14% of all actions) resulted from errors made by the speech recognition component. The DA classifier also causes 21.49% of non-eligible actions (corresponding to about 6% of all actions).

Wrong DA	Wrong Transcribed Text	Wrong Action by Agent
21.49%	46.73%	31.77%

TABLE VI: Contribution of each component to non-eligible actions taken by the HEL agent

All in all, we argue that the success rate of 92.86% is remarkable, the average number of turns of 8.38, and the average Likert score of 7.9 leads us to conclude that our RL framework for multimodal human-robot interaction performs exceptionally well.

VI. Conclusion

In conclusion, our work addresses the critical challenge of improving assistive robots to make a positive impact on the lives of older adults and people with disabilities. By focusing on the interaction manager, we've tackled the complexity of multimodal human-robot interactions. Our approach, including the novel neural network-based user simulator, leads to more effective and efficient robot training.

We've recognized the difficulties in obtaining realistic interaction data, especially when considering various modalities like language and haptic signals. Our data augmentation method helps bridge this gap, making our approach more accessible and valuable for the community.

Overall, our study shows that using reinforcement learning and a multimodal user simulator can make assistive robots better at their jobs. We hope our work will contribute to the development of more capable and helpful domestic assistive robots in the future.

Appendix

A. Background

The previous studies [1], [16] are based on the ELDERLY-AT-HOME corpus [2], a publicly available dataset containing records of human-human interactions in a home setting. These interactions revolve around assisting with daily activities like putting on shoes and cooking (ADLs). Specifically, HBATNs framework is built using a subset of this corpus that focused on a task known as the "Find" task. Our RL framework uses the same data. In this task, an elderly person (ELD) requested an object, and a helper (HEL) attempted to locate it by asking additional questions.

In [1], the "Find" task is broken down into a set of subtasks. The primary aim is to figure out two key pieces of information: the object you're looking for (O) and where it can be found (L). This breakdown involves four main subtasks, which are: determining the type of object desired (*Det(*O_T*)*), identifying a potential location to check (*Det(*L*)*), opening that location (*Open(*L*)*), and finally, determining the actual object (*Det(*O*)*). These subtasks are represented using Action-Transition Networks (ActNets).

The ActNet is a bipartite graph that represents the states of both participants (ELD and HEL) and the possible actions they can take using various forms of communication. These actions are defined by a combination of linguistic features (like the *dialogue act* (DA) [2], which reflects the speaker's intent and the words related to objects or locations) and physical features (such as pointing gestures or haptic-ostensive actions). The HBATNs framework incorporates these ActNets, enabling a robot not only to understand its partner's current state but also to plan its own actions accordingly.

In the subsequent paper [16], they expanded the model to allow the robot to take on the roles of either the elderly participant (ELD) or the helper (HEL). To achieve this, they broke down each subtask into smaller, more detailed sub-subtasks that is referred to as "primitive subtasks." In this revised approach, tasks like determining the object type (*Det(*O_T*)*) and identifying the location (*Det(*L*)*) were further divided into establishing the object type (*Estab*) and, optionally, confirming it (*Verify*), or asking follow-up questions for more details (*Spec*). Similarly, the task of determining the actual object (*Det(*O*)*) included confirming the presence or absence of the desired object (*Finish*) in the current location or verifying a physical object with the partner. This study demonstrated that the HBATN framework, coupled with a trained classifier to determine the current subtask participants are engaged in, can effectively model and perform both the roles of the helper (HEL) and the elderly participant (ELD).

In this study, we introduce a reinforcement learning (RL) approach to determine the best course of action in human-robot interactions. This RL policy replaces the manually designed HBATN policy for the helper (HEL) robot. As outlined in section I, a significant challenge during the RL training process is the need for an interactive environment

where the agent can learn. To address this challenge, we create and utilize a simple user simulator that takes on the role of the elderly participant (ELD) and interacts with the RL agent during the training process.

We used a user simulator in our RL training, which we named the ***Basic User Simulator Model (BUS Model)***. This simulator effectively conveyed the intended actions to the helper (HEL) agent, demonstrating the robustness of our RL framework even in the presence of imperfect interactions. However, enhancing the accuracy of such a user simulator remains a challenging research task. The next section of this paper discusses our efforts to create a more precise user simulator, which we refer to as the ***Generic User Simulator Model (GUS Model)***.

B. Tables

	NoUtt	Inst	Ack	Q-w	Q-yn	R-w	R-y	R-n	Chk	Exp	Algn	St-y	St-n	St
NoUtt	**67.79%**	0%	0%	0%	0%	1.69%	16.95%	0%	0%	0%	0%	6.78%	0%	0%
Inst	12%	**46%**	0%	0%	0%	12%	0%	**30%**	0%	0%	0%	0%	9.09%	0%
Ack	0%	0%	**66.67%**	0%	0%	0%	0%	0%	0%	0%	0%	**33.33%**	0%	0%
Q-w	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Q-yn	–	–	–	–	–	–	–	–	–	–	–	–	–	–
R-w	0%	2.32%	0%	0%	0%	**93.03%**	4.65%	0%	0%	0%	0%	0%	0%	0%
R-y	0%	0%	0%	0%	0%	0%	**100%**	0%	0%	0%	0%	0%	0%	0%
R-n	0%	20%	0%	0%	0%	0%	0%	**80%**	0%	0%	0%	0%	0%	0%
Chk	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Exp	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Algn	–	–	–	–	–	–	–	–	–	–	–	–	–	–
St-y	20%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	**80%**	0%	0%
St-n	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	**100%**	0%
St	–	–	–	–	–	–	–	–	–	–	–	–	–	–

TABLE B.1: DA Classification Confusion Matrix. Cells with – indicate that there were no such DA labels in the evaluation data set and the model did not predict any of such DAs.

	No Act	Give O_T	Give L	Give O_T, L	Ack	Yes	No
No Act	**74.08%**	0%	0%	0%	0%	25.92%	0%
Give O_T	8.82%	**69.12%**	0%	0%	0%	22.06%	0%
Give L	10%	0%	**83.33%**	0%	0%	6.67%	0%
Give O_T, L	–	–	–	–	–	–	–
Ack	0%	0%	0%	0%	**66.67%**	33.33%	0%
Yes	0%	3.12%	0%	0%	0%	**96.88%**	0%
No	0%	11.11%	0%	0%	0%	0%	**88.89%**

TABLE B.2: Action Classification Confusion Matrix. Cells with – indicate that there were no such Action labels in the evaluation data set and the model did not predict any of such Actions.

C. RL Framework Evaluation, User Interface Implementation

Since our framework is trained on multi-modal human data, it is capable of interpreting and executing multi-modal actions, and in particular language and physical actions. However, in this work, we focus on language and as a proof of concept, we extract other modalities from utterances. So the *Perceived Human Intent* vector that results from human subjects is limited to their utterances. The HEL agent is also capable of performing pointing and H-O actions in addition to communicating through language. However, in our evaluation, the agent generates the speech based on its predicted DA, and then also informs the human about the other modalities through speech. For instance, when the agent is verifying if the human uttered "drawer" with a pointing gesture it announces: *"The agent points to the drawer."*

For the human subjects to be able to have a smooth interaction with our agent, we developed and implemented a friendly user interface following the architecture for the Perception Module and the Execution Module in Fig. 1. Since the HEL agent is trained based on DA tags and the action vector generated by the user simulator, it expects to get DA tags and action vectors of the human subject in the test phase as well. As a result, we need to extract the DA tags and the action vectors from the human utterances. Therefore, we have Speech-to-Text, Action Extractor, and DA Classifier components which are explained in detail later on. And for the agent to respond back to the human we first need to transform its DA tag and action vector into a sentence and a textual description of the action, respectively. To this end, we developed a rule-based text generator inspired by the ELDERLY-AT-HOME corpus. Finally, the Pyttsx3 Python library [25] is used to transform the agent's sentences into speech.

Human speech is passed to the Google Cloud Speech-to-Text API [26] for speech recognition. Then, the transcribed text is passed to a DA classifier as well as an action extractor. We developed our DA classifier by deploying Sentence-BERT [27] and extracting embeddings from our data as the input to the classifier. We then trained a two-layer (one hidden layer followed by the output layer) feed-forward Pythorch neural network, with a dropout layer (ratio=0.1), and ReLU activation function after output layer. We used the gold-standard DA tags as the ground-truth labels and optimized the weights by employing the cross-entropy loss function and Adam optimizer. The accuracy of our Sentence-BERT-based DA classifier tested on 10% of ELDERLY-AT-HOME data is 67.23%. Our action extractor module works based on the objects and locations it extracts from human utterances. For example, if it detects one of the objects in the utterance, that's labeled as *"Give O_T"* action; if it detects one of the locations in the utterance, that's labeled as *"Give L"* action; if it detects one of the objects as well as one of the locations in the utterance, that's labeled as *"Give O_T, L"* action. For extracting these words of interest, we built a dictionary based on NLTK dictionary [28] consisting of all the words that could be pronounced similarly or close to our words of interest (to compensate for the speech recognition returning similarly sounding words that do not make sense in the context).

REFERENCES

[1] B. Abbasi, N. Monaikul, Z. Rysbek, B. Di Eugenio, and M. Žefran, "A multimodal human-robot interaction manager for assistive robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 6756–6762.

[2] L. Chen, M. Javaid, B. Di Eugenio, and M. Žefran, "The roles and recognition of haptic-ostensive actions in collaborative multimodal human–human dialogues," *Computer Speech & Language*, vol. 34, no. 1, pp. 201–231, 2015.

[3] I. Bratko, T. Urbančič, and C. Sammut, "Behavioural cloning: phenomena, results and problems," *IFAC Proceedings Volumes*, vol. 28, no. 21, pp. 143–149, 1995.

[4] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," *arXiv preprint arXiv:1805.01954*, 2018.

[5] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[6] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[9] B. Woodworth, F. Ferrari, T. E. Zosa, and L. D. Riek, "Preference learning in assistive robotics: Observational repeated inverse reinforcement learning," in *Machine learning for healthcare conference*. PMLR, 2018, pp. 420–439.

[10] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[11] A. L. Thomaz, G. Hoffman, and C. Breazeal, "Reinforcement learning with human teachers: Understanding how people want to teach robots," in *ROMAN 2006-The 15th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, 2006, pp. 352–357.

[12] B. Liu, G. Tur, D. Hakkani-Tur, P. Shah, and L. Heck, "Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems," *arXiv preprint arXiv:1804.06512*, 2018.

[13] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems: Recent advances and new frontiers," *Acm Sigkdd Explorations Newsletter*, vol. 19, no. 2, pp. 25–35, 2017.

[14] X. Li, Y.-N. Chen, L. Li, J. Gao, and A. Celikyilmaz, "End-to-end task-completion neural dialogue systems," *arXiv preprint arXiv:1703.01008*, 2017.

[15] H. W. Park, I. Grover, S. Spaulding, L. Gomez, and C. Breazeal, "A model-free affective reinforcement learning approach to personalization of an autonomous social robot companion for early literacy education," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 687–694.

[16] N. Monaikul, B. Abbasi, Z. Rysbek, B. Di Eugenio, and M. Žefran, "Role switching in task-oriented multimodal human-robot collaboration," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2020, pp. 1150–1156.

[17] L. Peternel, W. Kim, J. Babic, and A. Ajoudani, "Towards ergonomic control of human-robot co-manipulation and handover," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. Birmingham: IEEE, Nov. 2017, pp. 55–60.

[18] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, "Assistive gym: A physics simulation framework for assistive robotics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 169–10 176.

[19] S. Nikolaidis, J. Forlizzi, D. Hsu, J. Shah, and S. Srinivasa, "Mathematical models of adaptation in human-robot collaboration," *arXiv preprint arXiv:1707.02586*, 2017.

[20] L. El Asri, J. He, and K. Suleman, "A sequence-to-sequence model for user simulation in spoken dialogue systems," *Interspeech 2016*, pp. 1151–1155, 2016.

[21] F. Kreyssig, I. Casanueva, P. Budzianowski, and M. Gasic, "Neural user simulation for corpus-based policy optimisation of spoken dialogue systems," in *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, 2018, pp. 60–69.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[24] A. Mehri Shervedani, K.-H. Oh, B. Abbasi, N. Monaikul, Z. Rysbek, B. Di Eugenio, and M. Zefran, "Evaluating multimodal interaction of robots assisting older adults," *arXiv e-prints*, pp. arXiv–2212, 2022.

[25] [Online]. Available: https://pyttsx3.readthedocs.io/en/latest/

[26] [Online]. Available: https://cloud.google.com/speech-to-text

[27] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[28] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.