Group 10

Seyedarman Vaziri Bozorg

Matthew Coopman

Shruti Udupa

CSE 6363 – 007

Intelligent NPC Using NLP, LLM, and RL

I. Introduction

NPC stands for non-playable character, a form of artificial intelligence (AI) in gaming. NPCs help immerse the user/player into the game setting and/or help move along the game until it reaches an end state. Usually, NPCs are limited to pre-scripted dialogue and even actions. However, now that the field of AI is growing, more dynamic, humanlike dialogue/actions are expected nowadays. For example, ChatGPT is a prevalent implementation of AI and is also defined as a large language model (LLM). While ChatGPT can respond well to the user, it is not able to adjust to dynamic context. The problem is that LLMs cannot handle dynamic context and/or specific context (use case). A solution is to incorporate natural language processing (NLP) for user dialogue interpretation with respect to current conversation context and reinforcement learning (RL) for handling game state/context. For this implementation, the NPC will host a quiz game for the user. The NPC will select and ask questions appropriate to the current game state and respond to off-topic or related messages from the user.

II. Related Works

This application takes some inspiration from other transactional chatbot designs. Transactional chatbots are AI fixated on determining the user's goal, moving towards it through conversations, and finally performing the related actions. Similarly, this NPC focuses on providing questions of appropriate difficulty to the player, and if it sees that the player is struggling, it will provide a hint for a more comfortable experience. One transactional chatbot implementation could help a user go through the process of checking the availability for a movie and buying a ticket [1]. In fact, this one had a similar design to the NPC's. It used natural language understanding (NLU), a dialogue manager (DM), and a natural language generator (NLG). On the other hand, the NPC uses LLM in place of NLG and uses NLP and RL in place of NLU and DM. Nevertheless, the dialogue manager is an interesting concept. It ended up being used to hold context and serve as the natural language processing part in the current NPC.

In addition to research into other implementations, metrics regarding training/testing were considered. Since the NPC was a full system that used multiple models, end-to-end testing metrics became a valid concern. For instance, BLEU and ROUGE scores are metrics that other conversational models have used [2]. They compare the actual responses with the reference responses from the dataset based on string matching, word sequence, etc. However, these scores needed a dataset of user input and proper responses, and there was difficulty finding one involving a quiz game. Therefore, separate testing of the NPC's internal models was opted. Another concern was testing the reinforcement learning model because unlike other models, accuracy could not be easily calculated. Most metrics, such as reliability metrics proposed by Google researchers, focused on the learning curve of the reinforcement learning model [3]. Success rate was an option for reinforcement learning, but the term success in this non-deterministic game is vague, and there might be a lot of successful states. The decision ended up being to compare learning curves and other training data.

III. Methodology

i. Application Overview

The approach for this application is shown in Figure 3.1.1. As shown, the interface between the user/player and the NPC is a web application implemented with Python Flask. Through this interface, textual and auditory (NPC exclusive) information is passed. The NPC works with some external datastore, such as JSON Web API or file storage. For this implementation, we acquired question data in JSON format from a web API called Open Trivia Database and stored it as files for end-to-end testing [4]. These questions are used for messages sent to the user.

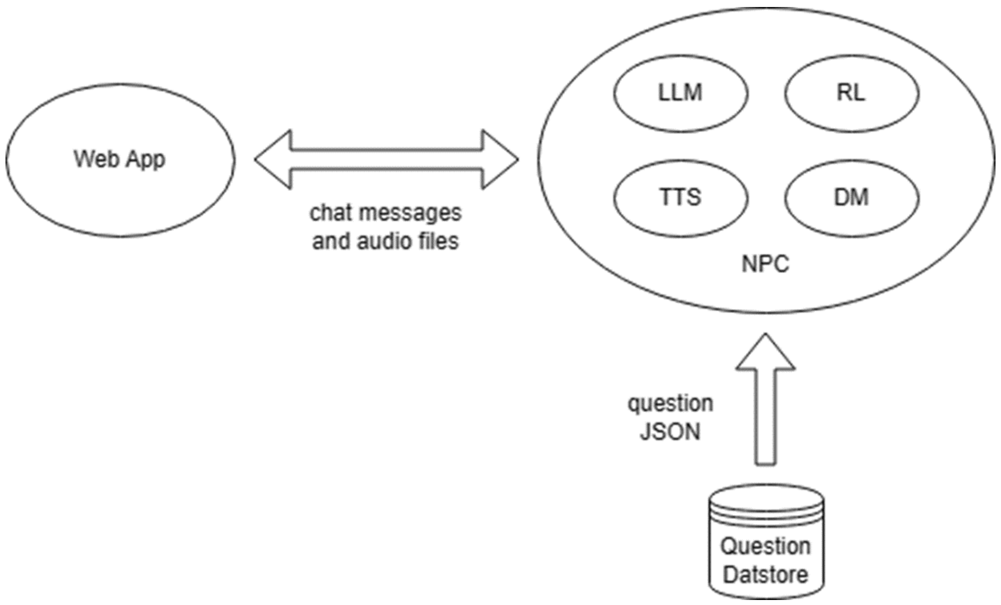


Figure 3.1.1. Application Diagram

The internal diagram for the NPC shown in Figure 3.1.2 consists of a dialogue manager (DM), a reinforcement learning model (RL), and a large language model (LLM). It does also contain an additional text to speech (TTS) feature. However, this feature will not be delved into much because it does significantly contribute to the workings of the NPC. Also, to note, the LLM is a pre-built model, so details regarding this model will not be discussed either besides configuration or its effect on the overall flow of the NPC.

As mentioned in the problem statement before, the NPC uses DM for managing context for both conversation and game, RL for game mechanics, and LLM for humanlike dialogue generation. When the NPC receives the user input, the input will be interpreted and tracked by the DM. Then, the DM determines whether the input deals with any kind of game state change and if so, passes current state to RL for the follow-up action. Once the action is determined from RL, the DM will update the current state and send simple response message along with the any related dialogue context to LLM and, in turn, TTS for both human-like textual/auditory responses.

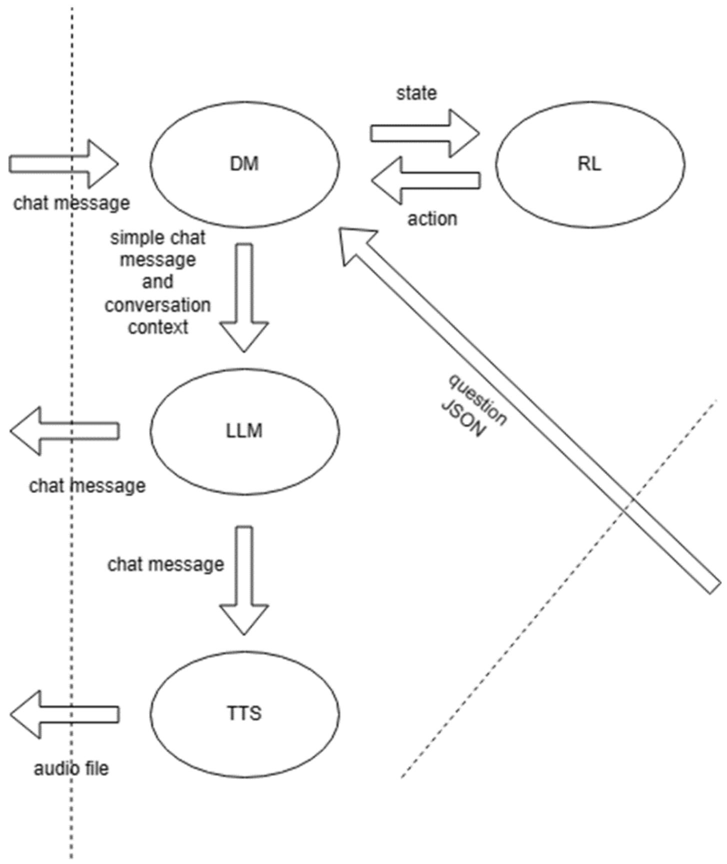
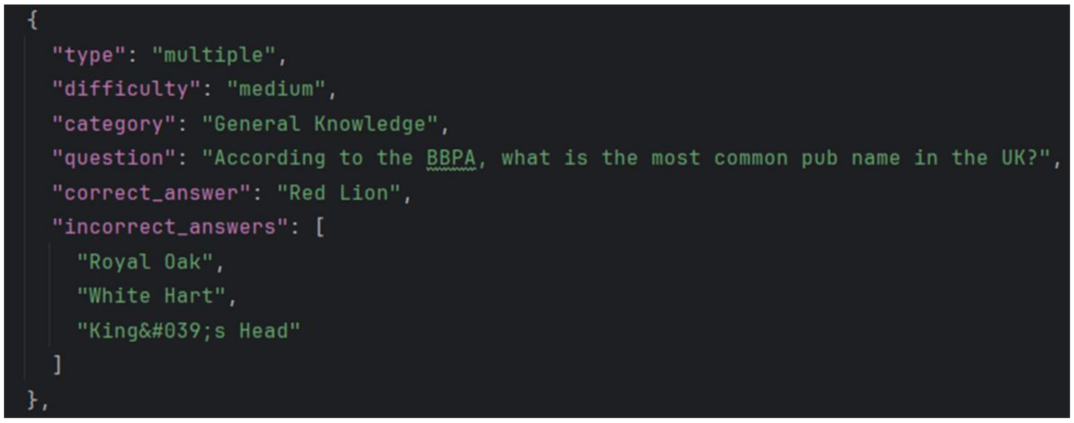


Figure 3.1.2. Internal NPC Diagram

ii. Dataset

The NPC uses three types of datasets: question dataset, user input dataset, and RL dataset. As shown in Figure 3.2.1, the question dataset is in JSON format. The features in this case are the JSON fields. Most of the fields are used except for “type” and “category” to not overcomplicate question selection for RL. Difficulty is the primary feature for question selection while the other fields pertain to game state. This dataset is not used for direct model training and testing, but it is important due to being part of the response messages to the user.



```json
{
  "type": "multiple",
  "difficulty": "medium",
  "category": "General Knowledge",
  "question": "According to the BBPA, what is the most common pub name in the UK?",
  "correct_answer": "Red Lion",
  "incorrect_answers": [
    "Royal Oak",
    "White Hart",
    "King&#039;s Head"
  ]
},
```

Figure 3.2.1. Sample Question JSON

These next two datasets are used for training and testing the models involved in the NPC. However, the datasets are not proper because they were created as the models were implemented. A proper dataset that pertained to the use case of a quiz game could not be found. The second dataset is the user inputs for intent classification. Intent classification is required for the DM to determine the current conversation context for further action. For instance, if the player was providing an answer to a previously mentioned question, the NPC needs to map that input to the user intent “giving_answer” so that the rest of the application can react accordingly. Table 3.2.1 shows the list of user intentions considered along with sample user input for each. Note that the number of user input samples used for this application is around 300.

Intent	Sample User Input
greetings	Hi there!
oos (out of scope)	I wanna talk to you about a different topic
affirm	Let's do this
decline	I'll pass.
affirm_ready	I'm ready to move on to the next question.
decline_ready	I'm not ready
giving_answer	The answer is 42.
need_hint	Hmm, I'm a bit unsure.
affirm_hint	Yes, I would like to use my hints.
decline_hint	I'd rather take a guess

Table 3.2.1. User Intent Dataset

The last dataset is the RL dataset. In the case of RL, the environment is the data that the model receives and manipulates to determine its output. Since the environment is a quiz game and the focus is hosting one, the action space needs to involve question selection, and the state space must contain metrics on the user/player's performance. Actual state/action space is detailed in Table 3.2.2 and Table 3.2.3. Based on these tables, the RL model will need to choose the best action for the state it is in. For example, the RL model should continue asking easy question (action "ask_easy") if the user has not answered many questions correctly (state "correct_answer_count"), and if the user misses too much (state "incorrect_answer_count"), the model should end the quiz game (action "end_game"). Within this quiz game environment, the model must learn how to host the game.

Action	Description
ask_easy	ask easy question and update state (difficulty) to easy
ask_medium	ask medium question and update state (difficulty) to medium
ask_hard	ask hard question and update state (difficulty) to hard
end_game	end game

Table 3.2.2. RL Action Space

State	Description	Type
difficulty	[easy, medium, hard]	discrete
incorrect_answer_count	number of incorrect answers (int)	continuous
correct_answer_count	number of correct answers (int)	continuous

Table 3.2.3. RL State Space

iii. LLM Configuration

The primary method for configuring a Language Model involves fine-tuning, a process that entails retraining a pre-existing LLM to align with our project's specific objectives. Fine-tuning adjusts the model's parameters through iterative training on task-specific data, allowing it to learn domain-specific nuances. However, fine-tuning poses computational challenges, particularly with large-scale LLMs such as the Llama model with 65 billion parameters [5]. Additionally, while fine-tuning enhances performance, it may not fully address the dynamic context issue inherent in language understanding tasks.

Alternatively, we can adopt a context injection [6] solution, where relevant context is directly incorporated into the model as a prompt. This approach bypasses the need for extensive retraining, leveraging learned context from intent classification and reinforcement learning components. By injecting context dynamically, the model can better adapt to varying contexts, improving overall performance in language understanding tasks.

In our project, we employed the GPT-3.5-turbo model [7] as the core LLM, utilizing OpenAI embeddings [8] to convert our question dataset into floating-point number vectors. We further utilized the LangChain library [9] to construct LLM chains, facilitating seamless integration of context into the model's inference process.

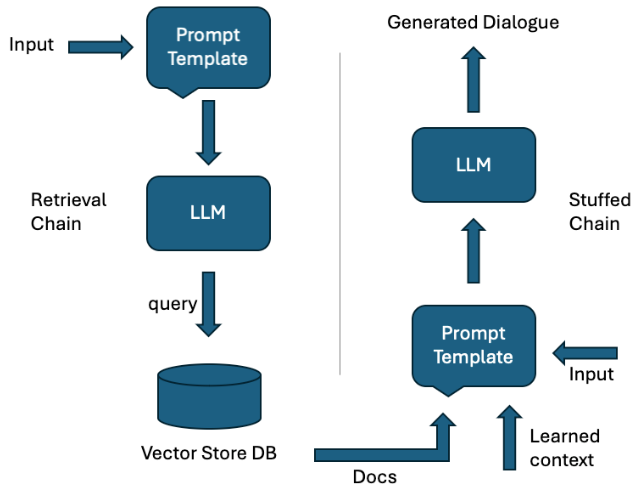


Figure 3.3.1. LLM Chains

As illustrated in Figure 3.3.1, our conversational chatbot employs two distinct chains to facilitate interaction. In the retrieval chain, the input, accompanied by a prompt template indicating the question's difficulty level (learned by the reinforcement learning (RL) component), is inputted into the Language Model (LLM). The LLM then generates a search query using cosine similarity search to retrieve a question from the vector store database that closely matches the input.

Subsequently, in the stuffed document chain, the retrieved question, user input, and context learned from the intent classifier embedded within the dialogue manager collectively serve as the prompt template for the LLM. This template guides the LLM in generating a response tailored to the user's query or input.

In the forthcoming section, we delve into the methodology employed for classifying user intentions, detailing the algorithms and techniques utilized to enhance the chatbot's understanding and responsiveness.

iv. Intent Classification Approach

To effectively handle dynamic context, the first step involves identifying the user's intention. Once determined, the appropriate context can be injected into the LLM chains. For instance, if the user's intention is classified as "affirm_ready," indicating readiness for the next question, the retrieval chain is activated to fetch a relevant question. This question then serves as the prompt template for the LLM to generate the output. Similarly, if the intention is classified as "giving answer," context is injected into the prompt template to provide appropriate feedback.

For training the intention classification model, we utilized a self-generated intent dataset, accessible in our GitHub repository. The model was trained using a deep sequential architecture comprising five layers: one Embeddings layer, one LSTM layer, two Dense layers, and one Dropout layer for regularization. Given the deep learning nature of the model, words and categorical labels were converted into vectors and padded with zeros to ensure uniform length. Additionally, regularization techniques were incorporated to enhance model generalization. Specifically, L2 regularization was applied to the LSTM and Dense layers, while a Dropout layer was inserted between the Dense layers to randomly eliminate features. The results, presented in the subsequent section, demonstrate the efficacy of our regularization approach, highlighting improved generalization compared to a simpler sequential model comprising only LSTM and Embeddings layers.

Finally, the trained model is saved, enabling the dialogue manager to utilize it for user intent classification and inject the appropriate context into the LLM prompt template.

v. Reinforcement Learning Approach

The dynamic context of our game can be effectively managed using Reinforcement Learning (RL) algorithms. These algorithms play a crucial role in enhancing specific game-related aspects within our Language Learning Model (LLM) chain. In our scenario, RL algorithms not only improve the overall game mechanics but also dynamically adjust the difficulty level of the questions posed by the Non-Player Character (NPC) to the user in the quiz game.

The intention to update the difficulty levels is triggered whenever the user is prepared for the next question during the game. We have integrated the Quiz Game environment with an RL model in a manner that adapts the difficulty level of the questions based on the user's responses. The game mechanics are designed such that the difficulty level remains constant when the user provides incorrect answers. However, if the user consecutively answers a certain number of questions correctly, the difficulty level is elevated. The game concludes after a set number of questions have been answered correctly or incorrectly.

This adaptive nature is incorporated into the Quiz Game environment to train RL models, enabling them to learn this environment and replicate it using the Q-Table. The details of the environment used are as follows:

1. Environment: The quiz game itself, where the NPC interacts with simulated user responses.
2. State Space: The number of correct and incorrect answers given by the user, and the current difficulty level of the questions being asked.
3. Action Space: Asking an easy, medium, or hard question, or ending the game.
4. Reward Function: Designed to motivate the Quiz master to increase the difficulty level when the user is answering correctly and maintain the level otherwise.

We have implemented two types of algorithms for the RL models:

1. Q-learning: An off-policy learning method, Q-Learning updates the Q-value for a certain action based on the reward obtained from the next state and the maximum reward from the possible states thereafter. It is considered off-policy as it employs an ε-greedy strategy for the initial step and a greedy action selection strategy for the subsequent step. The equation is shown in Equation 3.5.1.

$$Q(s_t, at_{)} \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Equation 3.5.1. Q-learning update rule

2. SARSA (State Action Reward State Action): An on-policy learning method, SARSA uses an ε-greedy strategy for all steps. It updates the Q-value for a certain action based on the reward obtained from taking that action and the reward from the state thereafter, assuming it continues to follow the policy. The equation is shown in Equation 3.5.2.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Equation 3.5.2. SARSA update rule

While training these models, certain hyperparameters are adjusted through trial and error to learn an optimal policy:

1. Gamma (γ): This is the discount factor used in the calculation of discounted future rewards. It determines how much importance we want to give to future rewards. A high value of gamma (close to 1) will consider future rewards significantly, while a low value will focus more on immediate rewards.
2. Epsilon (ε): This is used in the ε-greedy strategy for exploration during the learning process. A high epsilon value encourages more exploration, meaning the model will

take random actions to explore the environment. As the model learns, epsilon is reduced to allow the model to exploit what it has learned.

3. Alpha (α): Also known as the learning rate, alpha determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the latest information.

By fine-tuning these hyperparameters, we can ensure that our RL model learns effectively and efficiently, striking a balance between exploration and exploitation, and considering both immediate and future rewards. This results in an optimal policy that guides the NPC's actions in the quiz game.

After we train the RL models, we obtain the Q-table, which is then fed into the LLM chains. This triggers a check before the NPC is ready to ask a question to determine if there is a need to change the difficulty level or not.

IV. Results

i. Intent Classification Results

In Figures 4.1.1 and 4.1.2, we observe the model's performance metrics, including accuracy and loss, in comparison to the base model discussed earlier. The results indicate that the model trained on the dataset is overfitting, as evidenced by its high accuracy on the training data but inferior performance on the test data. Additionally, the model captures noise present in the training dataset.

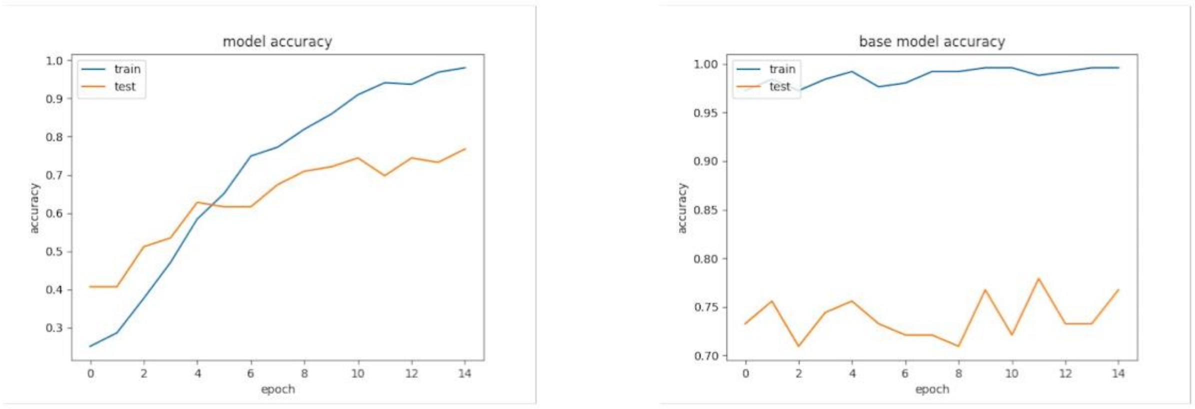
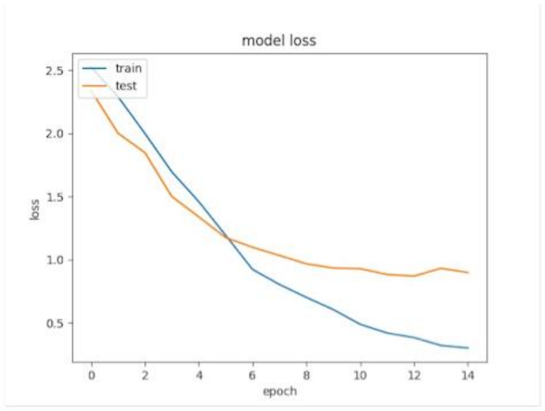


Figure 4.1.1. Accuracy on the datasets



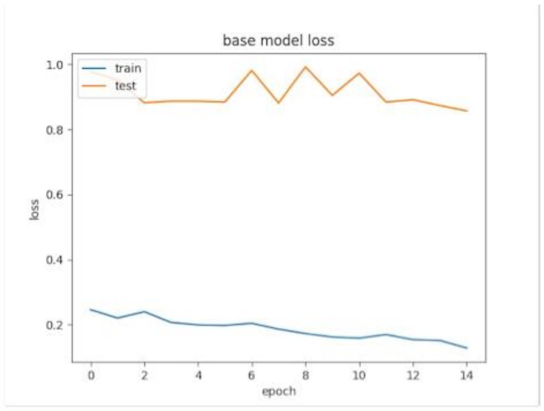
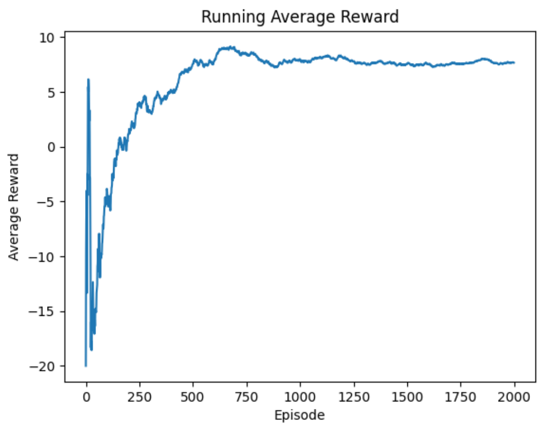


Figure 4.1.2. Loss on the datasets

Conversely, the proposed model exhibits fluctuations in accuracy and loss on the test data, suggesting potential issues with generalization. To achieve a more accurate model with better generalization capabilities, it is imperative to invest efforts in acquiring a more representative and reliable dataset. Alternative solutions to address this challenge are explored in the conclusion section.

ii. Reinforcement Learning Results

Running average rewards learning curves are a common method used to visualize the performance of an RL model over time. The y-axis represents the average reward, and the x-axis represents the number of episodes or iterations. At each step or episode, the agent receives a reward based on its action. These rewards are then averaged over a certain number of episodes to smooth out the curve and reduce the impact of short-term fluctuations or noise. If the curve is increasing, it indicates that the RL model is learning and improving its policy to achieve higher rewards. Conversely, a decreasing or fluctuating curve may suggest that the model is struggling to find an optimal policy. This visualization aided in understanding the learning process and in tuning the model's hyperparameters for better policy. In Figures 4.2.1 and 4.2.2, we can observe the learning curves of Q-learning and SARSA Models.



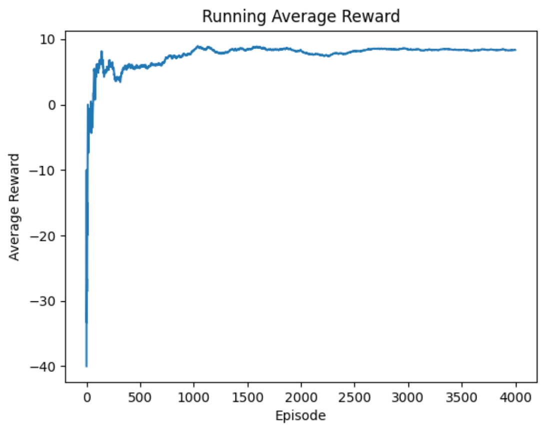


Figure 4.2.1. Q-learning learning curve

Figure 4.2.2. SARSA learning curve

Upon examining the learning curves of both Q-learning and SARSA, distinct patterns emerge that highlight their respective learning efficiencies. The Q-learning curve exhibits significant fluctuations, indicating a struggle in the model's ability to learn the optimal policy. This instability could be attributed to Q-learning's off-policy nature, which might lead to more exploration and less exploitation, causing the model to oscillate between different policies. On the other hand, the learning curve for SARSA presents a smoother trajectory, demonstrating a steady convergence towards the optimal policy. The rewards consistently increase over time, suggesting that SARSA is effectively learning from its experiences and refining its policy. This can be attributed to SARSA's on-policy approach, which ensures a balance between exploration and exploitation, leading to more stable learning.

Given these observations, we have chosen to utilize the Q-table derived from SARSA for integration into our LLM chain. This decision is driven by SARSA's demonstrated proficiency in consistently updating the difficulty level of questions in the quiz game, thereby enhancing the overall user experience.

V. Conclusion

One approach to enhancing the intent classifier model involves gathering a more diverse and comprehensive dataset. However, an alternative strategy is to leverage a Language Model (LLM) for intent classification [10]. Unlike traditional methods, the LLM-based classifier requires only a few examples per intent for training and exhibits rapid training capabilities. Moreover, it can be trained on multilingual data, enabling classification of messages in multiple languages. By employing an LLM for intent classification, we not only enhance classification accuracy but also unlock additional benefits.

The capabilities of the Language Learning Model (LLM) can be extended to inform a policy learner agent. This can be achieved by employing advanced techniques such as Deep Q-Networks (DQN), which can predict suitable actions corresponding to each identified intention within a larger state space. At present, our use of Q-learning or SARSA allows us to manage only a limited state and action space. This is due to the exponential growth of these spaces, which poses a challenge for traditional RL methods. However, this limitation can be effectively addressed by employing DQNs. DQNs, with their ability to handle larger state spaces and action spaces with ease, offer a promising solution. They leverage the power of deep learning to approximate the Q-value function, enabling the handling of complex problems with high-dimensional state and action spaces. This makes DQNs an excellent choice for enhancing the adaptability and performance of our system.

Regardless, if there were more time and resources, a more proper dataset would have been preferred over the current one. An option could be an extensive user input dataset for a different use case. While this sort of dataset would not normally fit this user case at hand, it could be treated as a superset-like dataset, and the user intent for that use case can be mapped to a subset that would help this application. Another option is transcriptions of quiz shows. If found/obtained, the user input would resemble more real-life scenarios and would be quite vast. Moreover, these transcriptions could potentially serve as reference responses for calculating BLEU and ROUGE scores for end-to-end testing.

In addition to considering DQNs, the NPC could utilize reinforcement learning in a different way. The NPC could consider user feedback as part of its reward process. That way, the NPC can better tailor questions for diverse users. Furthermore, if this kind of RL training is included, testing can be possible. For training and testing, different users could be simulated by providing the probability of answering a question correctly, measures that usually affect user feedback, etc.

To sum up, the development process of this NPC required good practices of both software engineering and machine learning. The resultant system needed to connect several types of models with the pre-built LLM to further enhance the quality of the interactions between user and NPC. The performance of these components was satisfactory based on base model comparisons and prior experience with RL learning curves. However, the NPC could benefit from additional improvements, such as proper dataset, RL alternatives, and more extensive LLM adoptions in other parts of the NPC. Still, this implementation serves as a good reference for further study in machine learning and user/machine interfaces.

References

[1] "How to train a transactional chatbot using reinforcement learning?" Accessed: Apr. 30, 2024. [Online]. Available: https://www.leewayhertz.com/train-transactional-chatbot-using-reinforcement-learning/

[2] Tran, Q.-D.L.; Le, A.-C. "Exploring Bi-Directional Context for Improved Chatbot Response Generation Using Deep Reinforcement Learning." *Appl. Sci*. 2023, 13, 5041. Accessed: May 3, 2024. [Online]. Available: https://doi.org/10.3390/app13085041

[3] Chan, Stephanie C.Y.; Fishman, Samuel; Canny, John; Korattikara, Anoop; Guadarrama, Sergio. "Measuring the Reliability of Reinforcement Learning Algorithms." ICLR, 2020. Accessed: May 3, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.1912.05663

[4] "Open Trivia Database." Accessed: May 3, 2024. [Online]. Available: https://opentdb.com/

[5] H. Touvron *et al.*, "LLaMA: Open and Efficient Foundation Language Models", Accessed: Apr. 30, 2024. [Online]. Available: https://github.com/facebookresearch/xformers

[6] "Build Your First LLM Chatbot. A beginner's guide to Large Language... | by Alisha Saboowala | Medium." Accessed: Apr. 30, 2024. [Online]. Available: https://medium.com/@alisha3/build-your-first-llm-chatbot-77456438f57b

[7] "Models - OpenAI API." Accessed: Apr. 30, 2024. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5-turbo

[8] "Embeddings - OpenAI API." Accessed: Apr. 30, 2024. [Online]. Available: https://platform.openai.com/docs/guides/embeddings

[9] "Vector store-backed retriever | LangChain." Accessed: Apr. 30, 2024. [Online]. Available: https://python.langchain.com/docs/modules/data_connection/retrievers/vectorstore/

[10] "Using LLMs for Intent Classification." Accessed: Apr. 30, 2024. [Online]. Available: https://rasa.com/docs/rasa/next/llms/llm-intent/