Testing Report for Bangla Chat Bot

The testing process for the Bangla Chat Bot focused on evaluating its ability to understand and respond accurately to user inputs in Bengali. Tests were designed to assess both functional and contextual understanding capabilities of the chatbot.

Test Methodology

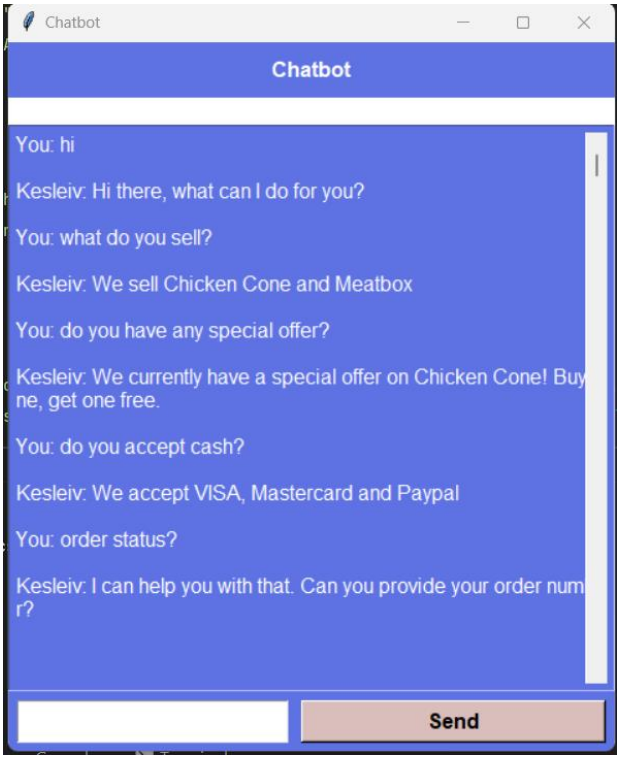
1. **Unit Testing**: Testing individual components (e.g., language detection, response generation) for functionality and accuracy.
2. **Integration Testing**: Ensuring all components (NLP processing, neural network model, GUI interaction) work together seamlessly.
3. **System Testing**: Evaluating the chatbot as a whole, focusing on its end-to-end functionality.

Test Cases

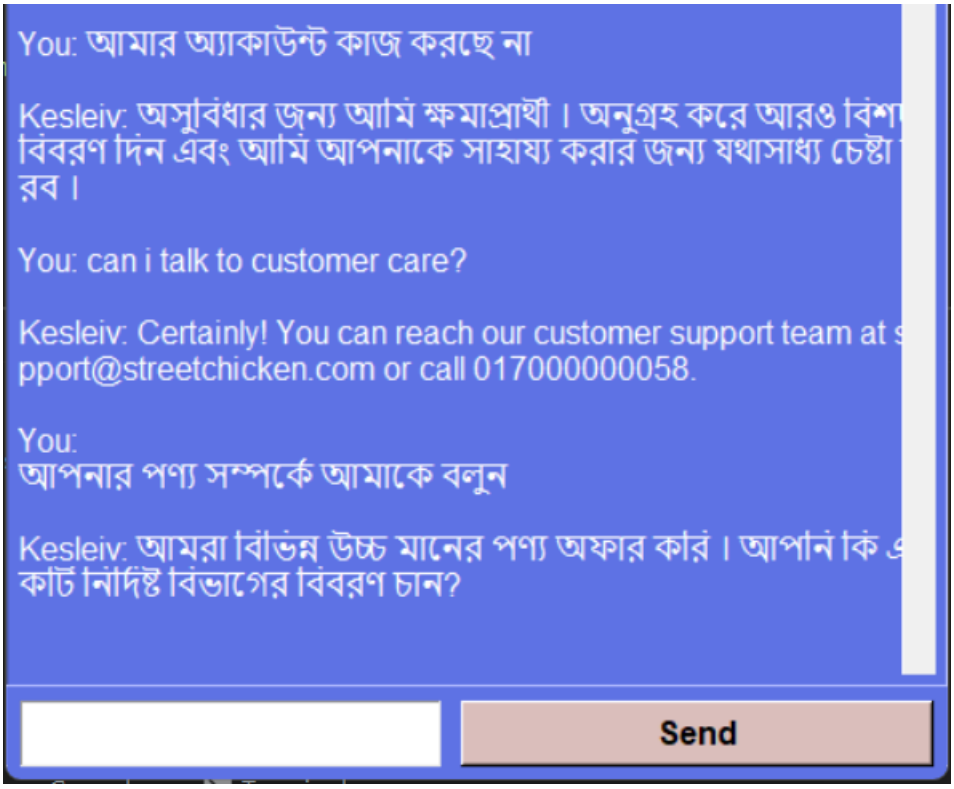
1. Language Understanding and Response Accuracy

- **Objective**: To test if the chatbot correctly understands both Bengali and English inputs and provides accurate responses.
- **Method**: Input predefined Bengali and English phrases/questions and evaluate the accuracy of responses.

Example Case:



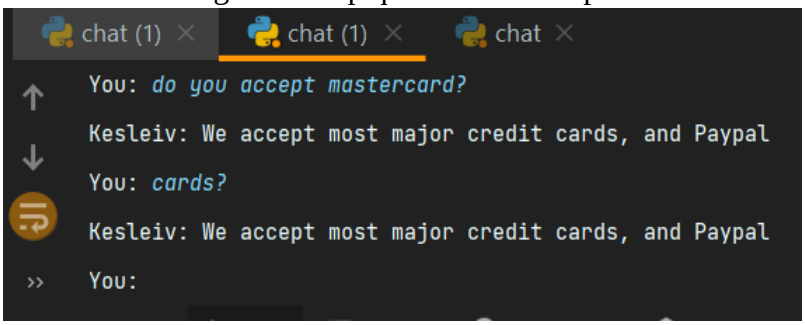
The testing results indicate that our Chatbot demonstrates a strong capability in classifying English inputs accurately and generating corresponding responses effectively. The precision of the responses is noteworthy, particularly considering that the user queries often deviate from the exact patterns provided in the training data. Despite these variations, the model adeptly identifies the underlying intents and responds appropriately, showcasing its robust understanding and classification abilities.



The chatbot is also identifying **Bangla** inputs and responding seamlessly.

2. Contextual Understanding

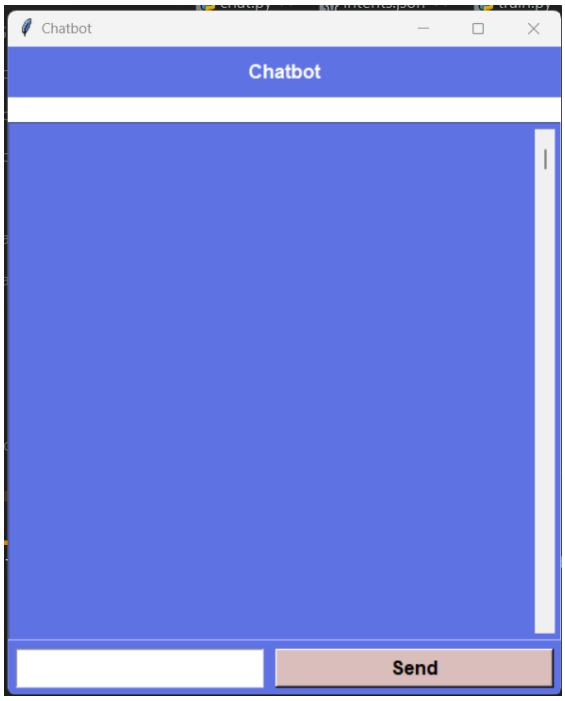
- **Objective**: To assess the chatbot's ability to maintain context in a conversation.
- **Method**: Engage in a conversation where the context from the initial query is essential for understanding follow-up questions. Example:



I have added context_set in the training dataset. The chatbot updates the context everytime user gives a known input. So when the user asked do you accept mastercard the context was “payment”, in the next question the question was “cards?” which is not a pre defined question but still the chatbot maintained the previous context and responded promptly.

3. GUI Functionality

- **Objective**: Ensure the GUI is responsive and user-friendly.
- **Method**: Interact with the chatbot through the GUI, testing different functionalities like input submission and response display.



This is our GUI. Its user-friendly a responsive.

Output of the training process for your chatbot's neural network model:

- **Patterns Processed**: The model was trained on 41 different patterns. These patterns are the sample inputs that correspond to various intents.
- **Tags Identified**: There are 12 unique intent tags. These are categories like 'customer_support', 'funny', 'goodbye', etc., that the model uses to classify inputs.
- **Stemmed Words**: The training process identified 85 unique stemmed words. Stemming reduces words to their root form, which helps in generalizing the learning process.
- **Training Epochs**: The training ran for 1,000 epochs, with the loss (a measure of the error the model makes) decreasing over time, which is a good sign. The loss values started at 1.1082 and dropped to 0.0005 by the last epoch, indicating the model's improving accuracy over the training period.

- **Test Accuracy**: After training, the model achieved a test accuracy of 33.33%. This is relatively low and suggests that the model may not generalize well to unseen data.

The test accuracy figure suggests there is room for improvement. To enhance the model's performance, consider the following steps:

- **Increase Data Diversity**: Add more varied patterns to each intent in **intents.json** to help the model learn a wider range of user inputs.
- **Hyperparameter Tuning**: Experiment with different model architectures, learning rates, or other hyperparameters.
- **Extended Training**: Sometimes, more epochs or a larger dataset can help improve accuracy.
- **Regularization Techniques**: Implement techniques like dropout to prevent overfitting if your training set is not large.
- **Post-Training Evaluation**: Use a separate validation dataset to evaluate the model's performance and prevent overfitting.

A low test accuracy despite a low training loss can indicate several issues, most commonly overfitting, where the model has learned to perform very well on the training data but fails to generalize to new, unseen data. Here are some potential reasons and considerations for this discrepancy:

Overfitting: If the model is too complex relative to the simplicity and amount of the training data, it may memorize the training examples instead of learning generalizable patterns.

Insufficient Validation: It's crucial to validate the model on a separate dataset that wasn't used during training. If the validation set is too small or not representative, the validation accuracy may not reflect the model's true performance.