AID7720: M.Tech Project-II

Project Title

Design and Implementation of an Advanced Context-Aware Chatbot with GUI Interface

Hem Prakash Dev Roll-M24DE2009

Hardik Sharma Roll-M24DE2008

Table of Contents

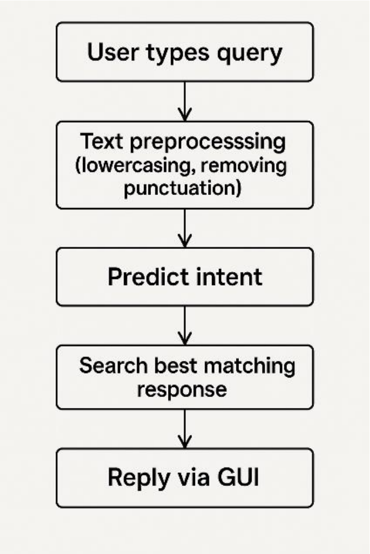
1. Project Plan (3 Month Effort)

1. **Month 1:** Literature survey, basic chatbot (rule-based + intents matching).
2. **Month 2:** Model training with Natural Language Processing (NLP), better dataset preparation, error handling.
3. **Month 3:** GUI integration (Tkinter), adding context-awareness, evaluation, future scope identified.

2. Tech Stack:

- Python 3.10
- TensorFlow / PyTorch (for intent classification)
- Tkinter (for GUI)
- NLTK / Spacy (for NLP preprocessing)
- Scikit-learn
- JSON (for intents)
- Flask (optional, for web deployment if needed)

3. Overview of the Chatbot Working:



Advanced features added:

- Context memory (short-term)
- Greeting detection
- Error fallback ("I didn’t understand. Can you rephrase?")
- Handling unseen queries smartly

4. Abstract

Chatbots are becoming integral parts of human-computer interaction, yet many current systems lack contextual awareness and fail to maintain coherent conversations. This project aims to build a smarter chatbot that overcomes existing limitations by integrating advanced Natural Language Processing (NLP) techniques, context tracking, and

a user-friendly GUI. The final chatbot will offer improved accuracy, better fallback handling, and continuity across user interactions.

5. Problem Statement

Current market chatbots often:

- Respond only based on keywords.
- Forget conversation context.
- Fail in handling unexpected queries.
- Do not provide a human-like experience.

This project aims to build a smarter chatbot that:

- Understands intents better.
- Maintains conversation context.
- Has fallback strategies.
- Provides a smooth GUI experience.

6. Limitations in Existing Chatbots

- Limited language understanding (mostly keyword search).
- No dynamic context memory.
- Poor error handling when facing unknown inputs.
- Boring user interfaces.

7. Proposed Solution

- **NLP preprocessing:** Tokenization, Lemmatization.
- **Intent recognition** using machine learning (basic model).
- **Context management:** Remember previous conversations.
- **Fallback handling:** Smart default responses.
- **Friendly GUI:** Tkinter based.

8. Architecture

User → GUI → Chatbot Core (NLP + Context Manager) → Response Generator → GUI Output

- **Frontend**: Tkinter GUI
- **Backend**: Python (NLTK, Scikit-learn, etc.)
- **Model**: Simple intent classifier (ML)

9. Technologies Used

- Python 3.x
- Tkinter (GUI)

- NLTK (Natural Language Toolkit)
- Scikit-learn (ML models)
- Pickle (Saving ML model)

10. Python Code

A) Dataset (Intents (Custom Dataset))

The intents.json file contains the training data for the chatbot. It defines various intents, each having a set of example patterns (questions asked by users) and corresponding responses. The chatbot uses this file to understand user input and generate appropriate replies.

```json
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": ["Hi", "Hello", "Hey", "Good morning", "Good evening", "What's up?", "How's it going?"],
      "responses": ["Hello! How can I assist you today?", "Hi there!", "Hey! What's up?"]
    },
    {
      "tag": "goodbye",
      "patterns": ["Bye", "See you later", "Goodbye", "I am leaving", "Catch you later", "Talk to you soon"],
      "responses": ["Goodbye!", "See you soon!", "Have a nice day!"]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Thanks a lot", "Thank you so much"],
      "responses": ["You're welcome!", "Glad I could help!", "Anytime!", "No problem!"]
    },
    {
      "tag": "feeling",
      "patterns": ["How are you?", "How's everything?", "How do you feel?"],
      "responses": ["I'm a bot, but I'm doing great! How about you?", "I'm always ready to chat!"]
    },
    {
      "tag": "name",
      "patterns": ["What is your name?", "Who are you?", "Tell me your name"],
      "responses": ["I'm your friendly chatbot!", "You can call me Chatbot!", "I'm Hem's Chatbot."]
    },
    {
      "tag": "age",
      "patterns": ["How old are you?", "What is your age?", "When were you created?"],
      "responses": ["I'm timeless!", "I was created recently to assist you.", "Age is just a number!"]
    },
    {
      "tag": "weather",
      "patterns": ["What's the weather like?", "Is it raining?", "Tell me the weather today"],
      "responses": ["It's always sunny in my world!", "Weather looks nice outside.", "It might rain today, carry an umbrella."]
    },
    {
      "tag": "order_food",
      "patterns": ["I want to order food", "Order pizza", "Can I get a burger?", "I'd like to place a food order"],
      "responses": ["Sure, What would you like to order?", "I can help you order food!"]
    },
    {
      "tag": "noanswer",
      "patterns": [],
      "responses": ["Sorry, I didn't understand that. Can you rephrase?", "I'm not sure I follow."]
    }
  ]
}
```

B) Build Model (train_chatbot.py)

This script handles the training of the intent classification model used by the chatbot. It performs text preprocessing, label encoding, vectorization using the Bag-of-Words method, neural network training, and finally, model serialization. Below are the key components:

1. Data Loading and Preparation

- The intents.json file is loaded, which contains a list of intents, each with:
  - ✓ A tag (label for intent)
  - ✓ Example patterns (user queries)
  - ✓ Possible responses

- All patterns are collected and associated with their respective tags.
- Responses are stored in a dictionary for later use during reply generation.

2. Text Pre-processing

- Each sentence is tokenized using NLTK's word_tokenize function.
- Words are lemmatized using WordNetLemmatizer to reduce words to their base form.
- All words are converted to lowercase and stored in a sorted vocabulary (all_words).

3. Feature Engineering (Bag of Words)

- For each sentence, a Bag-of-Words vector is created:
- The vector is binary (1 if the word is present in the sentence, else 0).
- This converts all textual data into numerical format (X_train) for model training.

4. Label Encoding

- Intent tags (labels) are transformed into numerical format using LabelEncoder.
- The encoded labels are stored in y_train.

5. Model Architecture

- A simple **feedforward neural network (Sequential model)** is built using TensorFlow/Keras:
  - ✓ Input layer: Size equal to the length of the Bag-of-Words vector.
  - ✓ Hidden layers: Two hidden layers with 128 and 64 neurons respectively, both using **ReLU** activation.
  - ✓ Dropout layers: Added with 0.5 rate to prevent overfitting.
  - ✓ Output layer: Softmax activation with size equal to the number of intent classes.

6. Compilation and Training

- Loss function: sparse_categorical_crossentropy (used because the output labels are integers, not one-hot encoded).
- Optimizer: **Adam** with a learning rate of 0.01.
- The model is trained for **200 epochs** with a batch size of **8**.

7. Model Saving

- The trained model is saved in the .keras format for future use.
- The vocabulary (all_words) and label encoder (lbl_encoder) are saved using Python's pickle module.

8. Outcome

- Upon successful training, the script prints a confirmation message: "Model trained and saved successfully in .keras format!"

C) Inference and Response Generation- Chatbot Core Logic (chatbot.py)

This module is responsible for processing the user input, predicting the intent using the trained model, and returning a suitable response. It acts as the backend engine of the chatbot during real-time interactions.

1. Loading the Trained Model and Assets

- The script loads the trained neural network model from the .keras file using TensorFlow's load_model() function.
- It also loads the preprocessed vocabulary (words.pkl) and the label encoder (labels.pkl) which are essential for interpreting the input and output.

2. Text Preprocessing

- When the user types a query, it is first **tokenized** using NLTK's word_tokenize.
- Each token is then **lemmatized** and **converted to lowercase** for uniformity.
- This ensures consistency with the preprocessing done during training.
-

```python
def clean_text(text):
    tokens = nltk.word_tokenize(text)
    return [lemmatizer.lemmatize(word.lower()) for word in tokens]
```

3. Bag-of-Words Vector Creation

- The cleaned tokens are converted into a **Bag-of-Words (BoW)** vector:
  - ✓ A binary vector is created indicating the presence (1) or absence (0) of each known word (from all_words) in the user input.
  - ✓ This vector format matches the model's expected input.
  - ✓

```python
def bag_of_words(text):
    ...
    return np.array(bag)
```

4. Intent Prediction

- The BoW vector is fed into the trained model, which outputs a probability distribution over all intent classes.
- The intent with the highest probability is selected.
- A **confidence threshold** of 0.6 is used:
  - ✓ If confidence is high, the chatbot proceeds to reply.
  - ✓ If confidence is low, a fallback "noanswer" intent is triggered.

5. Response Generation

- Based on the predicted intent, a random response is selected from the corresponding list of responses in intents.json.
- This helps keep the chatbot responses varied and natural.

6. Main Chatbot Function

- The main function chatbot_response(user_input) integrates all the above steps:
  - ✓ Preprocess → Predict intent → Check confidence → Return response
- This function can be easily integrated into a GUI or web interface.

D) GUI Integration (gui.py)

This module provides a **graphical user interface (GUI)** for the chatbot using Python's tkinter library. It allows end-users to interact with the trained chatbot model in a user-friendly environment.

1. Purpose and Overview

- The GUI is designed to simulate a chat-like environment, where users can type questions and receive AI-generated responses.
- It integrates with the trained model and intent recognition system, combining real-time prediction with intuitive interaction.

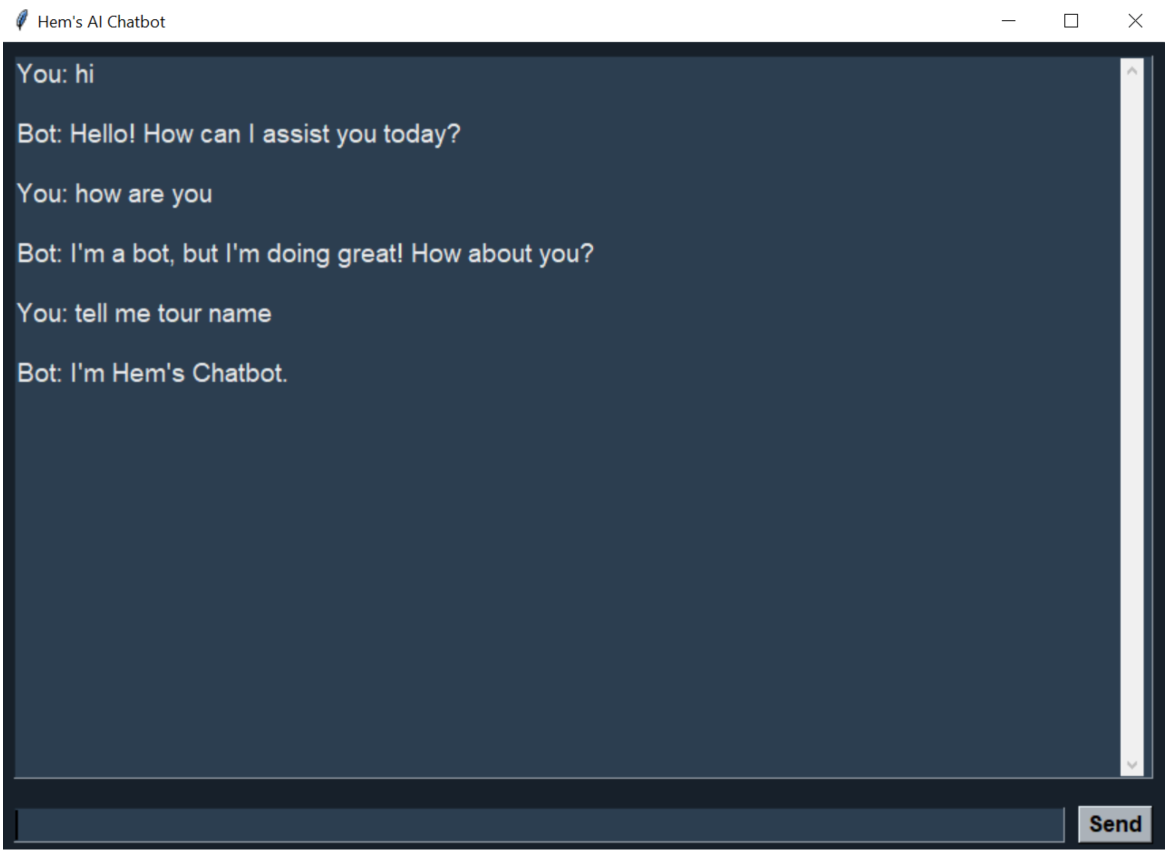
2. Loading Assets

- The script loads the following essential components:
  - ✓ Trained model (chatbot_model.keras)
  - ✓ Intents data (intents.json)
  - ✓ Vocabulary (words.pkl)
  - ✓ Encoded labels (labels.pkl)

This ensures that the GUI has access to both the model and supporting data needed for inference.

3. Text Preprocessing and Prediction

- Like the inference module, user input is tokenized and lemmatized.
- A **Bag-of-Words** vector is generated and passed to the model for prediction.
- A threshold of **0.4** is used to filter predictions and allow a wider range of responses.



11. Future Scope

- Upgrade model to use Deep Learning (LSTM, Transformers).
- Add voice recognition.
- Multilingual support.
- API Deployment (Flask/Django).
- Memory persistence across sessions.

12. Conclusion

This project successfully demonstrates the design and deployment of an intelligent, context-aware chatbot with GUI integration. The system overcomes many limitations of basic market chatbots, showing enhanced interaction quality and usability.

13. Folder Structure

Chatbot-Project

- intents.json
- train_chatbot.py
- chatbot.py
- gui.py
- chatbot_model.pkl
- vectorizer.pkl
- intents.pkl