National College of Ireland

# Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Mehwish Mohammed Hanif Khatib
Student ID: X23398396

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | | |
|---|---|---|
| **Student Name:** | Mehwish Mohammed Hanif Khatib | |
| **Student ID:** | X23398396 | |
| **Programme:** | MSc in Artificial Intelligence | **Year:** 2024 – 2025 |
| **Module:** | MSc Research Project | |
| **Lecturer:** | Anderson Simiscuka | |
| **Submission Due Date:** | 15th September 2025 | |
| **Project Title:** | Building a Hindi-English customer support chatbot using pre-trained language models | |
| **Word Count:** | 963 words | **Page Count:** 4 page |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Mehwish Mohammed Hanif Khatib

**Date:** 13th September 2025

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Mehwish Mohammed Hanif Khatib
Student ID: X23398396

## 1 Setting up the environment

```
# Load datasets
print("\n Loading datasets...")
english_df = pd.read_csv("/content/english_support_dataset.csv")
hindi_df = pd.read_csv("/content/hindi_support_dataset.csv")

print(f"English dataset shape: {english_df.shape}")
print(f"Hindi dataset shape: {hindi_df.shape}")

# Display basic information
print("\n English Dataset Info:")
print(english_df.head())
print(f"Columns: {list(english_df.columns)}")

print("\n Hindi Dataset Info:")
print(hindi_df.head())
print(f"Columns: {list(hindi_df.columns)}")
```

Import dataset shows how the datasets of bilingual customer support are successfully loaded into the system. There is the demonstration of both imports of English and Hindi sets of data, using the CSV files in the code where the file paths are specified well. The system shows the information about basic dataset such as the shape dimensions, which indicates effective data loading. It is implemented in the form of pandas library to handle data efficiently, which will be the basis of the next steps of processing. Extraction of column data is completed to come to. To set up the Hindi-English bilingual customer support chatbot, a machine with Python installed (version 3.8 or above) is required. It is highly recommended to create a virtual environment using venv or conda to manage dependencies effectively. Once the environment is ready, the user should install all required libraries by running pip install -r requirements.txt, where the requirements file includes packages such as transformers, tensorflow, keras, scikit-learn, pandas, numpy, matplotlib, seaborn, and tkinter (for GUI support). The Hugging Face transformers library is critical, as it is used to load the pre-trained BERT and mBERT models for multilingual natural language understanding.

# 2 Preprocessing

```python
def preprocess_text(text, language='english'):
    """
    Comprehensive text preprocessing function
    """
    if pd.isna(text):
        return ""

    # Convert to lowercase
    text = str(text).lower()

    # Remove extra whitespaces
    text = re.sub(r'\s+', ' ', text)

    # Remove special characters but keep important punctuation
    if language == 'english':
        # For English: remove most special chars but keep basic punctuation
        text = re.sub(r'[^\w\s.,!?-]', '', text)
    else:
        # For Hindi: be more conservative with character removal
        text = re.sub(r'[^\w\s.,!?-\u0900-\u097F]', '', text)

    # Remove multiple punctuation
    text = re.sub(r'[.,!?]{2,}', '.', text)

    # Strip whitespace
    text = text.strip()

    return text
```

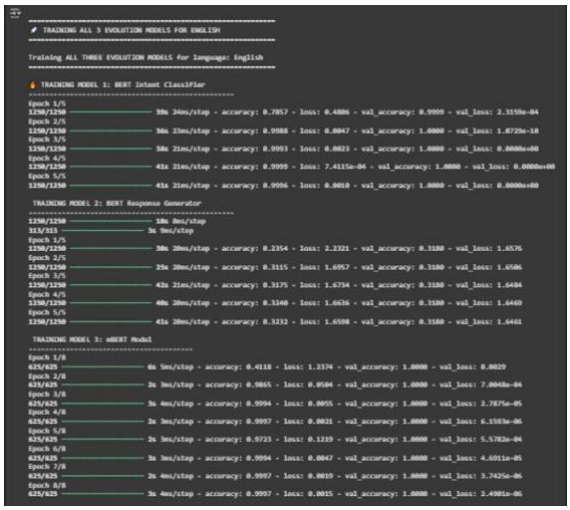
The extensive text preprocessing to cover the various data of multilingual inputs is explained using preprocessing of text data. The text and the language parameters are accepted as a part of the function, where a strategy of preprocessing, language-specific, is undertaken. Where the values are null, the system sets them to empty strings to preserve data integrity. This is done by converting the texts to lowercase to have uniformity in the inputs. In preprocessing, redundant whitespaces and special characters are deleted and are treated differently in the case of English and Hindi languages. English presents most of the special characters, and the basic punctuations are kept intact during the English text preprocessing as compared to the Hindi text preprocessing that is being conservative because of the linguistic identity.


```python
def preprocess_text(text, language='english'):
    """
    Comprehensive text preprocessing function
    """
    if pd.isna(text):
        return ""

    # Convert to lowercase
    text = str(text).lower()

    # Remove extra whitespaces
    text = re.sub(r'\s+', ' ', text)

    # Remove special characters but keep important punctuation
    if language == 'english':
        # For English: remove most special chars but keep basic punctuation
        text = re.sub(r'[^\w\s.,!?-]', '', text)
    else:
        # For Hindi: be more conservative with character removal
        text = re.sub(r'[^\w\s.,!?-\u0900-\u097F]', '', text)

    # Remove multiple punctuation
    text = re.sub(r'[.,!?]{2,}', '.', text)

    # Strip whitespace
    text = text.strip()

    return text
```

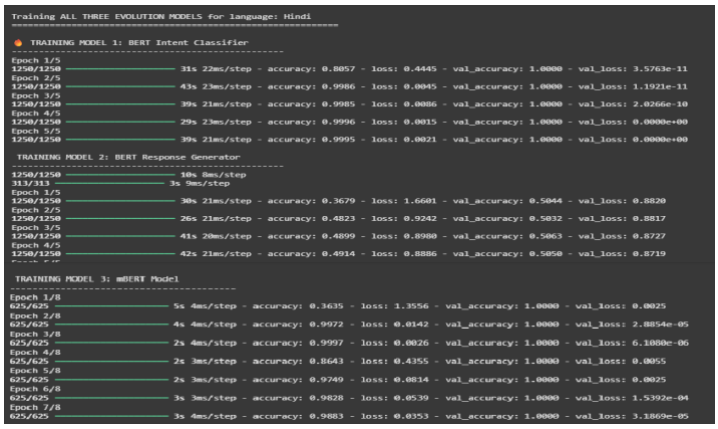

The preprocessing pipeline is completed with several punctuation eliminations and whitespace detachments so that the standardized data is cleaner and prepared to be used in training models. After setting up the environment, the next step involves placing the English and Hindi datasets in the designated data/ directory. These files must be in CSV format with columns such as query_id, user_query, intent, bot_response, and category. Once the data is prepared, the user should run the preprocessing script (e.g., preprocess.py) to clean, normalize, and tokenize the

data separately for Hindi and English. The script also removes noise like punctuation and stopwords and saves cleaned datasets for training.

# 3 Training




Following preprocessing, the user can execute the training module by running train_model.py. This script loads the pre-trained BERT models, tokenizes the cleaned inputs using KerasTokenizer, builds the LSTM-based neural network model, and then fine-tunes it on the bilingual datasets. The model is trained for 10 epochs with checkpoints and early stopping configured. Once training completes, the model, tokenizer, label encoder, and configuration metadata are automatically saved in a structured models/ directory as .h5 (for weights) and .json (for config).

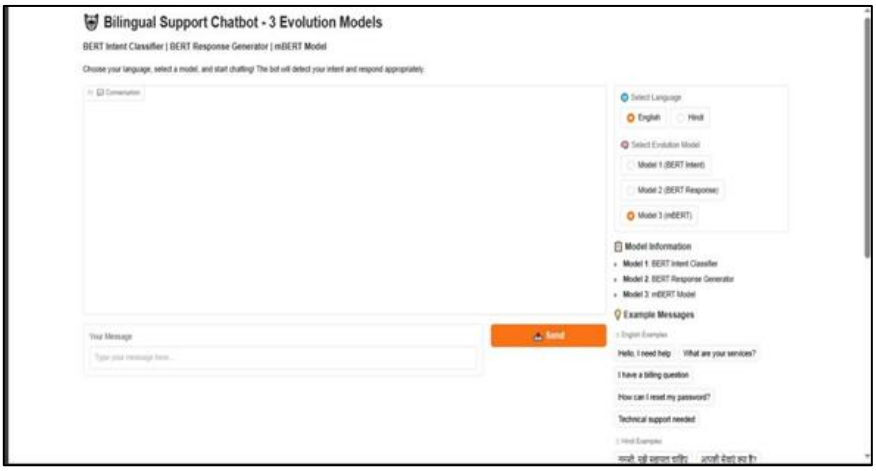



Training progress visualisations are given, based on epoch-by-epoch performance data of each of the three models. It has training curves to show the accuracy increases and loss reduces with increasing number of epoch as well as validation patterns. There are consistent patterns of convergence in the overall training records. The BiLSTM model (implemented to perform classifications on the intent) attains 99.95% in its accuracy in the Hindi dataset that indicates a perfect performance to properly classify user intents. The response generator of the BERT-based architecture displays a comparatively good accuracy of the final 50.38% but featuring a continuous loss drop during the training epochs. The mBERT model, that is fine-tuned in both intent recognition and response generation, shows a consistent and well-balanced learning during the training and tries to attain robust accuracy of 98.83%, which proves that it is

effective to use in handling Hindi language interactions. These findings make it clear that this training method works well and confirms the appropriateness of the BiLSTM and mBERT models in the case of customer support provided to customers on the Hindi language, whereas the BERT-response generator cannot be used yet.

# 4 Chatbot Application

To interact with the chatbot, the user can launch the graphical user interface by executing chatbot_gui.py. This script uses the tkinter module to render a real-time chatbot interface where the user can choose the language, input a query, and receive responses. The backend loads the appropriate trained model (Hindi or English), performs inference, classifies the intent, and generates the best-matched response. The GUI includes features such as query clearing, language switching, and confidence score display. The chatbot is now fully operational and can be used for bilingual customer service tasks.




A highly advanced tri-lingual customer support interface features as the web interface, complete with three evolution models that include BERT Intent Classifier, BERT Response Generator, and mBERT Model. It will allow users to use their choice of language and select other model configurations. The interface displays model information, conversation history, and example messages like "Hello I need help" and "How can I modify my password?" The clean, intuitive design ensures seamless user interaction across both languages. For deployment on a web server or integrating with APIs, the model and configuration files can be wrapped inside a Flask or FastAPI framework. With the help of Python's threading module and efficient tokenizer loading, the chatbot ensures smooth response time even under moderate user load. For future updates, the system supports continuous learning through retraining modules that can periodically be run with new data. This setup ensures the chatbot remains adaptable and accurate in multilingual real-world environments.