**Project Report: Automatic Ticket Classification using Many-to-One LSTM & Gemini Auto-Reply**

**1. Project Overview**

**Project Title:**  
Automatic Ticket Classification using Many-to-One RNN and Replied Back to Customer using LLM (Gemini API)

**Objective:**  
Organizations receive thousands of customer support tickets daily. Manual classification is error-prone, slow, and increases operational costs. The goal of this project is to:

1. Automatically classify customer tickets into their respective queues/departments.
2. Generate polite, generic replies acknowledging customer issues using a Generative AI API (Gemini 2.5 Pro).

**2. Skills & Technologies Learned**

* **Text Preprocessing & Tokenization**
* **Sequence Modeling using RNN / LSTM**
* **Model Evaluation Metrics:** Accuracy, Precision, Recall, F1-Score
* **Integration of ML with Generative AI (Gemini API)**
* **Prompt Engineering for Auto-Replies**

**Technical Stack:**

* **Programming Language:** Python
* **Libraries & Frameworks:**
  + **TensorFlow/Keras:** For LSTM-based sequence modeling
  + **NumPy, Pandas:** For data manipulation and processing
  + **scikit-learn:** For preprocessing and evaluation
  + **Hugging Face Datasets (Tobi-Bueck/customer-support-tickets):** Dataset source
  + **Streamlit:** For building the interactive web interface
  + **Gemini 2.5 Pro API:** For generating polite responses
  + **Seaborn & Matplotlib:** For visualization

**3. Problem Statement**

Customer support teams handle thousands of tickets daily. Misclassification leads to:

* **Delays in resolution**
* **Increased customer frustration**
* **Higher operational costs**

**Solution:**  
Automate ticket classification and auto-reply using LSTM for text classification and the Gemini API for generating polite responses. This reduces the human workload, improves ticket routing, and ensures customers receive timely acknowledgment.

**4. Business Use Cases**

1. **Customer Support Automation:** Automatically route tickets to the correct department (e.g., Billing, Technical Support, Account Services).
2. **Faster Ticket Resolution:** Reduce response times by providing pre-drafted acknowledgments.
3. **Cost Optimization:** Minimize manual triage effort, reducing operational costs.
4. **Customer Satisfaction:** Provide instant, empathetic replies to customers, improving their experience.

**5. Dataset**

* **Source:** Hugging Face — Tobi-Bueck/customer-support-tickets
* **Format:** JSON/CSV-like structure
* **Fields:**
  + **body** (Text of the ticket)
  + **queue** (Target department)
  + **priority, tags, subject** (optional)

**Explanation:**  
Each row in the dataset represents a customer support ticket.

* body is the input text describing the issue.
* queue is the target label, indicating the department to which the ticket should be assigned.

**6. Project Architecture**

**File Structure & Role**

| **File** | **Purpose** |
| --- | --- |
| data\_prep.py | Load dataset, preprocess text, tokenize, pad sequences, encode labels, split train/val/test, save artifacts. |
| model.py | Define LSTM architecture, save/load model functions. |
| train.py | Train model using artifacts, evaluate, save best/final model, generate confusion matrix. |
| predict.py | Load model + artifacts, predict queue for new tickets, optionally generate Gemini reply. |
| infer\_and\_reply.py | Alternative inference script, integrating Gemini API for auto-reply. |
| app\_streamlit.py | Interactive web UI to test ticket classification & reply generation. |
| utils.py | Helper functions (text cleaning, label decoding, logging). |
| requirements.txt | Python package dependencies. |
| README.md | Project documentation. |
| **Model Directory** | Trained model files (e.g., final\_model.keras). |

**Workflow Diagram:**

1. **Raw Ticket Data** (Hugging Face)
2. **data\_prep.py** → Preprocess & Save Artifacts
3. **train.py** → Load Artifacts → Build Model (model.py) → Train → Save Model
4. **predict.py / infer\_and\_reply.py** → Load Model → Predict Queue → Optionally Auto-Reply (Gemini)
5. **app\_streamlit.py** → Web UI for Interaction

**7. Data Preprocessing**

1. **Text Cleaning:**  
   Lowercase, remove punctuation, special characters, and non-printable characters.
2. **Tokenization:**  
   Convert text data into sequences of numeric indices using Keras Tokenizer.
3. **Padding/Truncation:**  
   Ensure all sequences have a fixed length (max\_len).
4. **Label Encoding:**  
   Convert categorical queue labels (e.g., Billing, Technical Support) into integer indices.
5. **Train/Validation/Test Split:**  
   Typically split data into 70% for training, 15% for validation, and 15% for testing.

**8. Model Architecture**

**Many-to-One LSTM:**

* **Embedding Layer:** Converts token indices to dense vectors.
* **Bidirectional/Unidirectional LSTM:** Learns sequential patterns in ticket text.
* **Dropout Layer:** Prevent overfitting.
* **Dense Layer (Softmax):** Output probability distribution over all queue classes.

**Hyperparameters:**

* **Embedding Dimension:** 128
* **LSTM Units:** 128
* **Dropout Rate:** 0.3
* **Max Sequence Length:** 100–200
* **Batch Size:** 64
* **Epochs:** 6–10

**9. Model Training**

* Uses **EarlyStopping** and **ModelCheckpoint** to save the best model based on validation accuracy.
* **Metrics Monitored:** Accuracy, Loss.
* **Evaluation:**
  + **Confusion Matrix** (saved as confusion\_matrix.png).
  + **Classification Report** (Precision, Recall, F1-score per class).

**Training Command:**

python train.py --epochs 6 --batch\_size 64

**10. Prediction / Inference**

1. **Load saved model** + **tokenizer** + **label encoder**.
2. Convert new ticket text to sequence & pad it to match the model's input shape.
3. Predict the target queue.
4. Optionally generate a polite reply using the **Gemini API**.

**Sample Code Snippet:**

ticket = "My payment failed during checkout"

label, probs = predict\_ticket(ticket, max\_len=100)

print(f"Predicted Queue: {label}")

**Example Output:**

Ticket: My payment failed during checkout

Predicted Queue: Billing

Probabilities: [0.85, 0.10, 0.05]

**11. Gemini Auto-Reply Integration**

* After predicting the queue, the ticket text + predicted queue are sent to the **Gemini API**.
* Gemini generates a polite, context-aware acknowledgment like:
* "We have received your issue regarding Billing, and our team will assist you shortly."
* This improves the customer experience by providing instant, empathetic responses.

**12. Results**

* The trained **LSTM model** achieves high accuracy in classifying tickets into their respective queues.
* **Confusion matrix** identifies any misclassifications between queues.
* **Gemini-generated replies** are polite and context-aware, ensuring high-quality customer interactions.
* The **Streamlit UI** demonstrates the end-to-end workflow, from classification to reply generation.

**13. Project Deliverables**

1. **Source Code**: Python scripts (train.py, model.py, etc.)
2. **Trained Models**: Saved models (saved\_models/final\_model.keras)
3. **Artifacts**: Tokenizer + Label Encoder files (tokenizer.pkl, label\_encoder.pkl)
4. **Documentation**: Detailed report and project documentation.
5. **Sample Outputs**: Example predictions and generated replies.
6. **Visualization**: Confusion matrix plot (confusion\_matrix.png).

**14. Evaluation Metrics**

| **Metric** | **Description** |
| --- | --- |
| **Accuracy** | % of correctly classified tickets. |
| **Precision** | Correct positive predictions / all predicted positives. |
| **Recall** | Correct positive predictions / all actual positives. |
| **F1-Score** | Harmonic mean of precision & recall. |
| **Confusion Matrix** | Visual inspection of misclassifications (e.g., misclassifications between queues). |
| **Reply Quality** | Subjective evaluation of the politeness & helpfulness of the generated replies (via Gemini). |

**15. Conclusion**

* **Successfully automated** ticket classification using a **Many-to-One LSTM**.
* Integrated **Generative AI** via **Gemini API** to generate polite, context-aware replies.
* The **end-to-end pipeline** reduces manual effort, response time, and enhances customer satisfaction.
* **Future Extensions**: The system can be extended to handle multi-label tickets, predict ticket priority, and be deployed in real-time helpdesk systems for live ticket processing.

**Appendices**

**A. Requirements**

**To run the project, ensure the following dependencies are installed:**

**pip install -r requirements.txt**

**The requirements.txt file includes the following major packages:**

* **TensorFlow/Keras: Deep learning framework for model building and training.**
* **Streamlit: Web framework for building the interactive UI.**
* **NumPy, Pandas: Essential libraries for data handling.**
* **scikit-learn: For preprocessing, metrics, and model evaluation.**
* **Seaborn & Matplotlib: Visualization tools for generating plots.**
* **Gemini API Client: For integrating the Gemini 2.5 Pro API for auto-replies.**

**B. Folder Structure and Key Files**

**project\_directory/**

**│**

**├── data\_prep.py # Preprocesses the dataset and saves artifacts**

**├── model.py # Defines LSTM model architecture and functions**

**├── train.py # Model training and evaluation script**

**├── predict.py # Predicts queue for new tickets**

**├── infer\_and\_reply.py # Generates auto-replies using Gemini API**

**├── app\_streamlit.py # Streamlit web UI**

**├── utils.py # Helper functions**

**├── requirements.txt # List of Python dependencies**

**├── saved\_models/ # Directory for saving trained models**

**├── tokenizer.pkl # Keras tokenizer for preprocessing text**

**└── label\_encoder.pkl # Label encoder for queue labels**

**C. Example Output**

* **Confusion Matrix: A heatmap that shows the true vs predicted classes. This helps visualize the performance of the classifier and where it may be making errors in classification.**

**Example Confusion Matrix (for three queues: Billing, Technical Support, and Account Services):**

**Billing Technical Support Account Services**

**Billing 95% 3% 2%**

**Technical Support 5% 90% 5%**

**Account Services 2% 4% 94%**

* **Sample Prediction & Reply:**

**Input Ticket:**

**"I cannot access my account, can you help?"**

**Predicted Queue:**

**"Technical Support"**

**Generated Reply (via Gemini API):**

**"We have received your issue regarding Technical Support, and our team will assist you shortly."**

**F. Acknowledgments**

**This project was inspired by the increasing demand for automated customer support systems and generative AI's potential to enhance the customer experience. Special thanks to the developers of the Gemini 2.5 Pro API and the Hugging Face community for providing the datasets and tools to build the foundation of this system.**

**G. References**

1. **TensorFlow Documentation -** [**https://www.tensorflow.org/**](https://www.tensorflow.org/)
2. **Keras Documentation -** [**https://keras.io/**](https://keras.io/)
3. **Streamlit Documentation -** [**https://streamlit.io/docs/**](https://streamlit.io/docs/)
4. **Hugging Face Datasets -** [**https://huggingface.co/datasets**](https://huggingface.co/datasets)
5. **Gemini 2.5 Pro API Documentation - (link to API docs, if available)**

**This comprehensive project report provides a clear understanding of the system architecture, goals, results, and potential future work for automating customer support ticket classification and auto-reply using LSTM and Generative AI.**