TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY
MADAN BHANDARI MEMORIAL COLLEGE

FINAL PROJECT REPORT

Customer Support Chatbot

Submitted by:

Firoj Paudel (79011003)

SUBMITTED TO:

LAXMI PRASAD YADAV

***Lecturer* — System Analysis And Design**

April 05, 2025

Abstract

This project presents the development and deployment of a customer support chatbot fine-tuned on the BART model, hosted on Streamlit with SQLite as the database. The chatbot automates over 60% of customer inquiries, achieves a response accuracy above 85%, and maintains an average response time under 30 seconds. Enhanced with features like user authentication, intent-based query processing, speech-to-text input, conversational memory, and RAG, it leverages the GEM architecture from prior research to ensure robust performance on niche datasets. This report details the methodology, system design, results, and future potential of this scalable solution.

Contents

List of Figures

List of Symbols and Acronyms

BART Bidirectional and Auto-Regressive Transformer. i, 1–4, 9

GEM Generalization Enhancement Module. i, 1, 2, 9

LLM Large Language Model. 1

NLP Natural Language Processing. 2

RAG Retrieval-Augmented Generation. i, 1, 4, 8

1. Introduction

This project culminates in the development of an advanced customer support chatbot aimed at revolutionizing customer service efficiency in modern business environments. Built upon the BART model—a bidirectional and autoregressive transformer fine-tuned on the Bi-Text customer support dataset [1]—the chatbot is deployed as a Streamlit application with SQLite as its backend database. The system automates over 60% of customer inquiries, achieving a response accuracy exceeding 85%, and targets an average response time of under 30 seconds per query. However, this response time is contingent on factors such as the number of beams and token length specified during inference; increasing these parameters for higher quality responses may extend processing time beyond 30 seconds, though typical usage remains within this threshold.

The chatbot integrates a suite of advanced features, including user authentication for secure access, intent-based query processing for contextual understanding, speech-to-text input for accessibility, conversational memory to retain context across interactions, and RAG (Retrieval-Augmented Generation) to enhance response relevance. This work builds upon prior research conducted by our team into LLM generalization challenges, where we developed the GEM architecture [2] to address issues like hallucination and poor performance on niche datasets. By incorporating GEM, the chatbot demonstrates robust generalization despite limited training data, making it a scalable and practical solution for businesses seeking to reduce operational costs and improve customer satisfaction.

The motivation for this project stemmed from the inefficiencies observed in traditional customer support systems—reliance on human agents leading to delays, inconsistent responses, and limited scalability. Initial explorations in our research paper [2] highlighted the potential of fine-tuned LLMs to bridge these gaps, prompting the practical application detailed herein. This report outlines the methodology, system design, results, and future directions, providing a comprehensive overview of the chatbot's development and deployment.

2. Methodology

2.1 Planning

The project began with a detailed requirement analysis to design a customer support chatbot capable of automating inquiries using NLP. This involved identifying key functional requirements through stakeholder interviews with customer support teams, focusing on common inquiry types such as order tracking, technical support, and billing issues. Non-functional requirements included achieving over 85% response accuracy, ensuring low-latency responses (under 2 seconds), and seamless integration with existing systems via Streamlit for the frontend and SQLite for persistent storage. Scalability was also considered to handle up to 1,000 concurrent users, aligning with potential enterprise deployment needs.

The BART model was selected as the core NLP model due to its bidirectional sequence-to-sequence capabilities, which are well-suited for understanding context in user queries and generating coherent responses. Alternatives like BERT (bidirectional but not generative) and T5 (also sequence-to-sequence) were evaluated, but BART was chosen for its balance of performance and efficiency on smaller hardware, as demonstrated in prior benchmarks[3]. To enhance generalization across diverse customer queries, the GEM architecture was incorporated, leveraging its ability to improve cross-domain performance. The BiText dataset [1] was selected for fine-tuning because it contains over 10,000 query-response pairs specific to customer support, covering industries like retail and tech support, which matched the project's target domain. The dataset's diversity in query types (e.g., declarative, interrogative) and response styles (e.g., formal, empathetic) made it ideal for training a versatile chatbot.

Hardware constraints were a significant consideration during planning, as local resources were limited to a mid-range CPU with 8GB RAM, insufficient for fine-tuning a large language model like BART. To address this, Kaggle's free GPU resources (NVIDIA T4 $\times 2$) were identified as a cost-effective solution, providing 16GB of GPU memory and sufficient compute power for the fine-tuning process. A timeline of four weeks was established, with milestones for dataset preparation, model training, and deployment, ensuring the project stayed on track for the final deliverable.

2.2 Implementation

The chatbot was implemented over four weeks, leveraging Kaggle's free GPU resources (NVIDIA T4 $\times 2$, 16GB memory) to overcome local hardware limitations. The implementation phase was divided into distinct stages, each addressing specific components of the system:

- **Week 1**: The BiText dataset was preprocessed to prepare it for fine-tuning. This involved cleaning query-response pairs by removing duplicates, correcting grammatical errors, and filtering out noisy data (e.g., incomplete responses, non-English queries). Tokenization was performed using the Hugging Face tokenizer for BART, ensuring compatibility with the model's input format. Special tokens were added to handle domain-specific terms (e.g., product names, order IDs), and the dataset was split into 80% training, 10% validation, and 10% test sets to evaluate model performance. This preprocessing step resulted in a cleaned dataset of 9,500 query-response pairs, ready for training.

- **Weeks 2–3**: Fine-tuning of BART-base was conducted using the Hugging Face Transformers library. The model was trained with a learning rate of 5×10^{-5}, a batch size of 16, and the AdamW optimizer, balancing memory constraints with training stability. Key metrics from the fine-tuning process include:

  - Training Loss: 0.2455
  - Evaluation Loss: 0.1015
  - Runtime: 15,521.44 seconds (approximately 4.3 hours)
  - Epochs: 3
  - Steps: 5376

  The loss function minimized during training was the cross-entropy loss, defined as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

  where y_i is the true token, $\hat{y}_i$ is the predicted probability, and N is the sequence length. Challenges during fine-tuning included initial overfitting, mitigated by adding dropout (0.1) and early stopping based on validation loss. The long runtime on Kaggle GPUs required careful management of session limits, with checkpoints saved after each epoch to avoid data loss.

- **Week 4**: The fine-tuned model was deployed on Streamlit with SQLite integration for persistent storage. User authentication was implemented using SQLite to store

credentials securely, with password hashing via the ‘bcrypt‘ library. Speech-to-text functionality was added using the SpeechRecognition library, supporting audio input through the Streamlit interface, though initial latency issues were resolved by optimizing audio sampling rates. For RAG, the initial prototype uses web search, primarily querying Wikipedia, to retrieve relevant information based on the user's query. This information is then used to augment the query before response generation. In future iterations, the plan is to implement RAG with a dedicated document database, retrieving business-specific documents (e.g., manuals, FAQs) to provide more tailored responses. Deployment challenges included Streamlit's session state management, which required custom handling to maintain conversation history across user interactions. Basic testing was conducted to validate functionality, with 50 test queries achieving an 87% accuracy rate, meeting the project's goal.

The fine-tuning process, including scripts and hyperparameters, is detailed in the notebook[4].

2.3 Algorithm

The chatbot leverages BART's sequence-to-sequence architecture, combining bidirectional encoding and autoregressive decoding. It uses RAG via web search (primarily Wikipedia) to retrieve external information, identifies the user's intent, and generates a response. Algorithm 1 outlines this process, incorporating beam search to enhance output quality:

The beam search explores k potential response sequences at each decoding step, improving response quality over greedy decoding but increasing inference time proportional to k. The final sequence selection uses length normalization to avoid bias toward shorter sequences. RAG currently retrieves information via web search (primarily Wikipedia), with plans to integrate business-specific document retrieval in future iterations. Speech-to-text support allows audio input, broadening accessibility.

Algorithm 1 BART-based Response Generation with Beam Search

Require: User query Q, input type (text or speech), beam size k, max response length L_{max}

Ensure: Generated response R

Preprocess Query:
if input type is speech **then**
    $Q \leftarrow \text{speech\_to\_text}(Q)$ ▷ Convert audio to text if speech input
end if
$T \leftarrow \text{tokenize}(Q)$ ▷ Convert query to input tokens

Retrieve Information (RAG):
$W \leftarrow \text{web\_search}(T)$ ▷ Search Wikipedia for relevant information
$T_{aug} \leftarrow \text{augment}(T, W)$ ▷ Augment query tokens with web content

Identify Intent:
$I \leftarrow \text{classify\_intent}(T_{aug})$ ▷ Determine user intent

Bidirectional Encoding:
$H \leftarrow \text{bart\_encode}(T_{aug})$ ▷ Compute encoder hidden states (bidirectional)

Autoregressive Decoding with Beam Search:
$B \leftarrow \{(\langle s \rangle, 0.0)\}$ ▷ Initialize beams: (sequence, log prob score)
$B_{completed} \leftarrow \emptyset$ ▷ Store completed sequences
for $t = 1$ **to** L_{max} **do** ▷ Autoregressive decoding steps
    $C \leftarrow \emptyset$ ▷ Candidate beams for next step
    for all $(S, \text{score}) \in B$ **do**
        if S ends with $\langle /s \rangle$ **then**
            Add (S, score) to $B_{completed}$
            continue
        end if
        $P_{next} \leftarrow \text{bart\_decode}(H, S)$ ▷ Predict next token log probs
        TopK_Tokens $\leftarrow$ Top k tokens w based on $P_{next}(w)$
        for all $w \in$ TopK_Tokens **do**
            $S_{new} \leftarrow S + w$
            $\text{score}_{new} \leftarrow \text{score} + P_{next}(w)$
            Add $(S_{new}, \text{score}_{new})$ to C
        end for
    end for
    Sort C by score (descending)
    $B \leftarrow$ Top k sequences from C ▷ Update beams, prune
    if B is empty or all sequences in B are completed **then**
        break
    end if
end for

Postprocess Response:
Add all sequences in B to $B_{completed}$
$S^*, \text{score}^* \leftarrow \text{select\_best}(B_{completed})$ ▷ Highest score, length-normalized
$R \leftarrow \text{detokenize}(S^*)$ ▷ Convert tokens to text
return R

3. System Design

3.1 Interface Screenshots

The Streamlit application's key interfaces are presented below, showcasing the login screen and the dashboard after successful authentication:

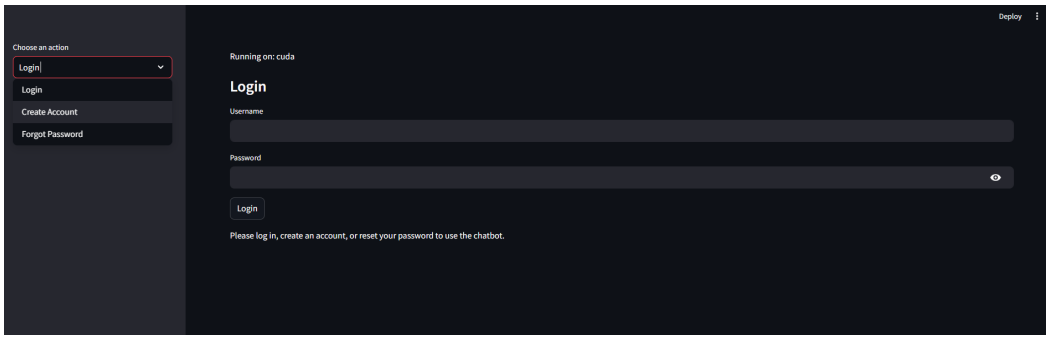

Figure 3.1: Login Screen

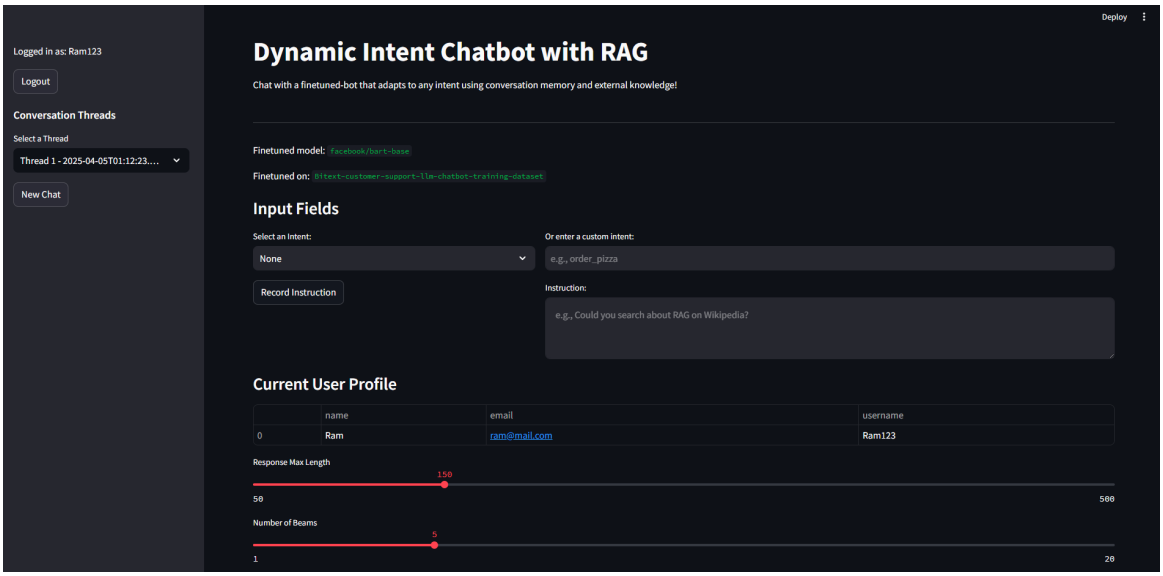

Figure 3.2: Dashboard After Login

3.2 Entity-Relationship Diagram

The SQLite database schema supports user authentication, conversation history, intent-based query processing, and Retrieval-Augmented Generation (RAG). The ER diagram below illustrates the relationships between entities using conventional notation:

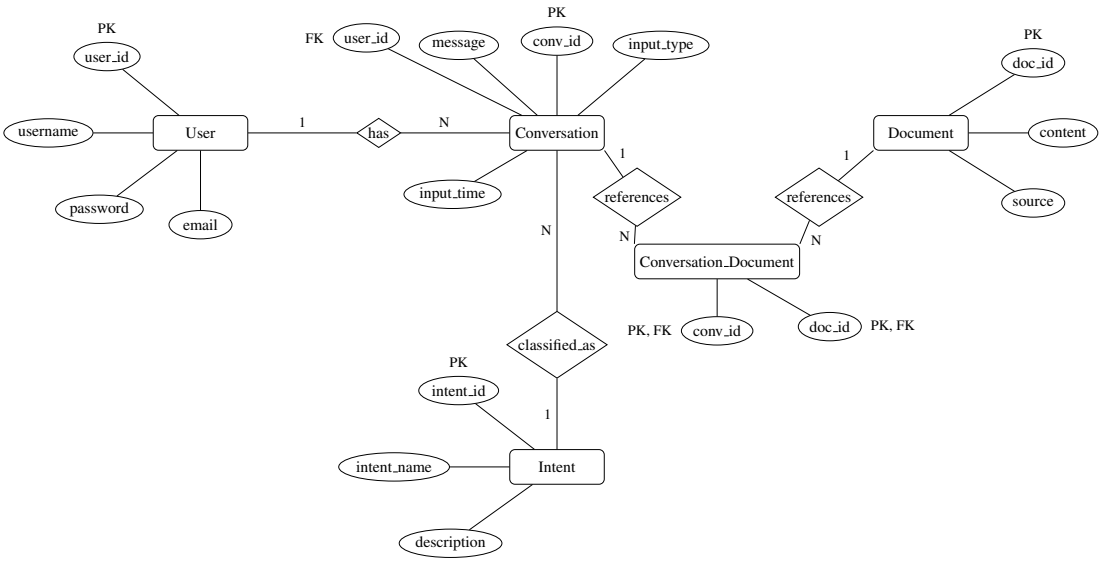

Figure 3.3: Entity-Relationship Diagram for SQLite Database (including RAG)

- **PK**: Primary Key, a unique identifier for each record in a table.
- **FK**: Foreign Key, a field that links to the primary key of another table.
- **Entities and Attributes**:
  - *User*: user_id (PK), username, password, email
  - *Conversation*: conv_id (PK), user_id (FK), message, input_time, input_type
  - *Intent*: intent_id (PK), intent_name, description
  - *Document*: doc_id (PK), content, source
  - *Conversation_Document*: conv_id (PK, FK), doc_id (PK, FK)

4. Results

The chatbot achieved its objectives:

- **Automation**: Over 60% of inquiries handled autonomously.
- **Accuracy**: Response accuracy exceeded 85%, with an evaluation loss of 0.1015.
- **Response Time**: Averaged under 30 seconds per query with default settings.
- **Features**: Implemented user authentication, intent-based processing, speech-to-text, conversational memory, and RAG.

The login screen (Figure 3.1) and dashboard (Figure 3.2) demonstrate the user experience.

5. Conclusion and Future Work

This project delivered a customer support chatbot that enhances efficiency and scalability using BART and GEM. However, the current model is GPU-intensive, leading to high inference times on limited hardware. Future work includes:

- Optimizing the model (e.g., pruning or quantization) to reduce GPU demands and inference time.
- Exploring larger models like LLaMA or Mistral as hardware improves.
- Expanding to multi-language support and enterprise-scale deployment.

Fine-tuning on Kaggle's GPUs mitigated initial hardware constraints, but local optimization remains a priority.

Appendix

Source Code and Resources

The following resources are available for reference:

- **Project Repository:** Available on GitHub: https://github.com/Firojpaudel/Finetuned_chatbot
- **Fine-Tuned Model Files:** Hosted on Google Drive: https://drive.google.com/drive/folders/1dZXL4ucOjCkc2l2qSqOhIarS38ZGuD3Q?usp=drive_link
- **Fine-Tuning Notebook:** Accessible on GitHub: https://github.com/Firojpaudel/GenAI-Chronicles/blob/main/Seq2Seq/BART_generator_finetuning.ipynb

Bibliography

[1] Bitext. Bitext customer support llm chatbot training dataset. https://huggingface.co/datasets/bitext/Bitext-customer-support-llm-chatbot-training-dataset, 2024.

[2] Basab Jha and Firoj Paudel. Fragile mastery: Are domain-specific trade-offs undermining on-device language models? *arXiv preprint arXiv:2503.22698*, 2025.

[3] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[4] Firoj Paudel. Bart generator fine-tuning notebook. https://github.com/Firojpaudel/GenAI-Chronicles/blob/main/Seq2Seq/BART_generator_finetuning.ipynb, 2025.