HONOURS - AML IA-1

A PROJECT REPORT

on

# “QUESTRAG: A Banking QUEries and Support system via Trained & Reinforced RAG”

By

**Group Number 9**

**Members:**

* Rohit Deshpande: 16010122041
* Eeshanya Joshi: 16010122074
* Minav Karia: 16010122083

B.Tech VII Semester Computer Engineering, KJ Somaiya School of Engineering, Mumbai

## Chapter 1: Introduction

#### 1.1 Overview

The rapid digitalization of banking services in India has led to the widespread adoption of chatbots to enhance customer support. However, existing banking Chatbots often fail to meet user expectations due to their reliance on rigid, rule-based systems. These systems struggle with informal or real-world queries, leading to customer dissatisfaction and reliance on traditional support channels. This project aims to develop an advanced banking chatbot to provide context-aware, conversational responses. By addressing the limitations of current systems, the proposed solution seeks to improve user satisfaction, reduce operational costs, and streamline banking support. This initiative aligns with the growing demand for efficient, user-centric digital solutions in India’s banking sector, offering a scalable approach to enhance customer experience.

#### 1.2 Challenges with Existing Systems

Banking chatbots in India, despite their widespread adoption, fall short in delivering effective customer support, often leaving users dissatisfied. This section outlines the critical limitations of current systems, highlighting the need for a more robust solution.

* **Limited Response Flexibility**: Current banking chatbots rely on hardcoded, rule-based responses, restricting their ability to handle informal or nuanced user queries effectively.
* **Inadequate Handling of Real-World Queries**: Systems often fail to address real-life scenarios, leaving users frustrated and unable to resolve complex issues.
* **Predefined Query Categories**: Chatbots typically offer rigid category selections, limiting user interaction and failing to accommodate diverse or unique problems.
* **Low User Satisfaction**: Due to ineffective query resolution, many customers remain dissatisfied, leading to distrust in automated systems.
* **Dependence on Traditional Channels**: Inefficient chatbot performance forces users to resort to time-consuming customer support calls or physical bank visits.
* **Lack of Contextual Understanding**: Existing systems struggle to interpret context, resulting in irrelevant or incomplete responses to user inquiries.
* **Scalability Issues**: Current chatbots are not equipped to handle high volumes of varied queries, impacting their reliability during peak usage.

Addressing these challenges is crucial to enhancing customer experience and operational efficiency in the banking sector. A more intelligent, context-aware chatbot system can bridge these gaps, reducing reliance on traditional support channels and improving user satisfaction.

#### 1.3 Objectives

The primary goal of this project is to develop an advanced banking chatbot system that overcomes the limitations of existing solutions, offering seamless and intuitive customer support. The following objectives guide this initiative:

* **Domain-specific query handling**

Build an LLM-powered assistant tailored for resolving real-world Indian banking queries using RBI regulations and policy-based retrieval.

* **Token cost optimization**

Integrate a reinforcement-learned policy model to decide when to retrieve external context, reducing unnecessary token usage during LLM inference.

* **Response Evaluation via Reward Signals**

Use an automated output evaluator to rate LLM responses and guide future decisions through reward-driven learning.

* **Output Guardrail with Reward-Based Correction [16]**

Deploy an output guardrail that evaluates each LLM-generated response against predefined ground rules, assigns a reward based on quality, and improves subsequent responses by feeding this reward into the reinforcement learning loop.

These objectives aim to revolutionize banking customer support by delivering a user-centric, efficient, and accessible chatbot system, setting a new standard for digital banking interactions.

#### 1.4 Contributions

This project introduces a transformative approach to customer support in the banking domain, addressing critical gaps in existing chatbot systems. Its contributions enhance both consumer redressal and operational efficiency in banks.

* Designed and implemented a Retrieval-Augmented Generation (RAG) system fine-tuned for Indian banking queries, combining dense document retrieval with a Groq-based LLM to generate grounded, regulation-aware responses using RBI policy data.
* Trained a BERT-based policy network using reinforcement learning to optimize context-fetching decisions (FETCH vs NO\_FETCH), effectively reducing token consumption by minimizing unnecessary document retrievals during inference.
* Developed a reinforcement learning pipeline where an evaluator assigns rewards to LLM outputs based on correctness and usefulness, enabling iterative improvement of the policy model via policy gradients.
* Integrated an automated guardrail **[16]** that evaluates generated responses against domain-specific constraints, assigns rewards accordingly, and closes the loop by informing subsequent policy updates—ensuring the chatbot learns to avoid hallucinations and follow predefined quality criteria over time.

By addressing these areas, the project significantly improves consumer support, fosters trust, and enhances efficiency, paving the way for a more responsive and inclusive banking ecosystem.

## Chapter 2: Literature Survey

This literature survey synthesizes key resources relevant to the development of a RAG + RL based consumer support system. This chapter is organized into three subsections: RBI circulars for regulatory compliance, research papers for advanced technical methods, and GitHub repositories for practical implementations. These resources collectively ensure the chatbot is compliant, secure, and efficient in addressing customer queries.

#### 2.1 RBI Circulars

The Reserve Bank of India (RBI) circulars provide a regulatory framework for banking operations, ensuring compliance, security, and customer-centric services. These circulars are critical for aligning the banking chatbot with Indian financial regulations, covering grievance redressal, IT governance, cybersecurity, and customer service standards.

* **Strengthening of Grievance Redress Mechanism in Banks (RBI/2020-21/87, January 27, 2021)** **[9]**

According to this circular, a chatbot must retrieve and provide accurate information on grievance procedures, along with customer support as well. Its logging mechanism should support compliance by tracking any possible discrepancies for disclosure and systemic analysis. Integration with CMS should enable seamless handling of digital complaints, reducing escalations. The chatbot ensures transparency by providing clear, policy-compliant responses, with a goal to align as much as possible with the circular’s policies. This circular ensures the chatbot supports efficient and transparent grievance redressal, a core requirement for banking customer service.

* **Master Direction on Information Technology Governance, Risk, Controls and Assurance Practices (November 7, 2023, effective April 1, 2024)** **[10]**

This circular establishes IT governance standards, requiring oversight by an IT Strategy Committee, third-party risk management, and cybersecurity measures like vulnerability assessments and audit trails. According to this circular, the chatbot must adhere to secure development practices, ensuring robust management of APIs and AI models. It should be Compliant with data protection laws to safeguard sensitive customer information is necessary. Regular assessments should be added while scaling this project to production level, which will protect the chatbot from security risks. The circular’s emphasis on continuity plans ensures that the chatbot remains operational during disruptions.

* **Cyber Security Framework in Banks (RBI/2015-16/418, June 2, 2016)** **[11]**

This circular establishes a cybersecurity framework, including a Cybersecurity Policy, Security Operations Centre (SOC), and incident response plans. The chatbot must be monitored by an SOC to detect threats like phishing. Integration with a Cyber Crisis Management Plan must be there to ensure prompt breach reporting to the RBI. The chatbot can educate users on cybersecurity practices, enhancing security. Regular vulnerability assessments must be done to ensure that the chatbot meets regulatory security standards.

* **Master Circular on Customer Service in Banks (RBI/2015-16/59, July 1, 2015)** **[12]**

This circular consolidates customer service standards, emphasizing grievance handling, financial inclusion (e.g., Basic Savings Bank Deposit Accounts), and transparency in service charges. According to this circular, the chatbot must provide access to bank policies (e.g., KYC, fees), enhancing service delivery. The chatbot must ensure transparency by sharing relevant details / sources with the user. It could aid vulnerable groups by offering simplified interfaces or voice-based assistance.

#### 2.2 Research Papers

Research papers provide advanced methodologies to enhance the banking chatbot’s RAG and RL systems, improving accuracy, efficiency, and reliability in handling financial queries. Below are the research papers referred for the project:

* **Optimizing Retrieval Augmented Generation for Domain-Specific Chatbots with Reinforcement Learning [1]**

Presented at AAAI 2024, this paper optimizes RAG for domain-specific chatbots (e.g., credit card FAQs) using RL, introducing an in-house embedding model and an RL policy for FETCH/NO\_FETCH decisions, achieving 31% token savings and improved accuracy. This is the main research paper which has been inspired to be used in the project.

* **Evaluating BERT-based Rewards for Question Generation with Reinforcement Learning [2]**

The paper explores the effectiveness of using BERT-derived reward signals in training question generation models via reinforcement learning. The authors compare various reward functions, including BERTScore and BLEU, to assess how well they guide the learning process. Experiments show that BERT-based rewards yield more fluent and context-relevant questions compared to traditional metrics. The study highlights the limitations of token-level rewards and supports semantic-level evaluation in RL-based question generation.

* **Self-Reasoning for Retrieval-Augmented Language Models [3]**

The paper proposes a novel method that enhances RAG performance by integrating a self-reasoning mechanism into the generation process. The model iteratively reflects on retrieved content and questions before generating a final answer, improving factual accuracy and coherence. The approach enables the language model to better utilize retrieved evidence and refine its responses autonomously. Experimental results show significant gains across multiple open-domain QA benchmarks. This method demonstrates a scalable way to boost retrieval-augmented generation without external supervision.

#### 2.3 GitHub Repositories

GitHub repositories offer practical implementations and tools to develop and refine the banking chatbot, focusing on RAG, RL, and financial applications.

* **RL-Self-Improving-RAG [4]**

This repository implements an RL-enhanced RAG system using FAISS and Groq, optimizing document retrieval for question-answering tasks. The system architecture used in the repository could match with the chatbot’s RAG and RL systems. RL policy optimizes retrieval decisions, reducing resource usage. Logging supports regulatory requirements. At the end, a blueprint for chatbot development is provided. This repository is a direct resource for building the chatbot, aligning with its technical architecture.

* **ARENA [5]**

ARENA repository improves RAG performance using RL with adaptive rewards, focusing on multi-hop question-answering and transparency. Response Quality can be Enhanced by utilizing the a judge LLM for better responses. Traceable logs and other data helps enhance the system. Complex Queries which help to handle multi-hop banking queries effectively. Adaptability: Principles can be applied to the chatbot’s LLMs. This repository complements the chatbot’s systems, improving response quality and compliance.

* **RAG\_Techniques [6]**

A comprehensive resource for advanced RAG techniques, including graph-based retrieval and query transformations. Retrieval Accuracy can be enhanced by improving document retrieval with advanced methods by referring to this repository. Evaluation Tools: DeepEval as used in this repository, can assess chatbot performance. Practical Resources: Notebooks aid implementation. This repository adopts methods that support ongoing chatbot refinement.

* **Financial-RAG-From-Scratch [7]**

A custom RAG system for financial queries, leveraging large financial datasets. This repository can be directly applicable to banking queries. It helps to enhance the chatbot’s financial dataset. It also offers insights for tailored RAG development.

* **Bridge\_the\_GAP [8]**

The Bridge the Gap Model (BGM) is proposed to enhance the performance of Retrieval-Augmented Generation (RAG) systems by addressing the preference gap between retrievers and language models (LLMs). In existing systems, the independent operation of retrievers and LLMs often leads to mismatches between retrieved content and the generation needs of the LLM.

In this work, BGM is introduced as a novel auto-regressive model designed to select, reorder, and adapt retrieved information to better align with LLM preferences. The model is implemented to process the retriever outputs along with the user query and return a filtered, ordered sequence optimized for LLM generation. Supervised training is currently employed, with optimized sequences obtained via a greedy search algorithm.

Performance improvements over standard RAG baselines are demonstrated through experiments on public Question Answering and custom generation datasets, highlighting the effectiveness of aligning retrieved content with the linguistic patterns required by LLMs.

#### 2.4 Context and Project Need

The project aims to develop a sophisticated banking chatbot tailored for Indian financial regulations, leveraging Retrieval-Augmented Generation (RAG) and Reinforcement Learning (RL) to provide accurate, efficient, and compliant responses to customer queries. The RAG system retrieves relevant documents from a knowledge base to generate context-aware answers, while the RL system optimizes the chatbot's decision-making process to balance efficiency and accuracy. This is particularly crucial in the Indian banking sector, where customers require precise information on complex regulations, and institutions need scalable, compliant solutions. By combining these advanced AI techniques, the project addresses the need for a reliable, adaptive, and resource-efficient banking assistant that can handle diverse queries while maintaining regulatory standards.

## Chapter 3: Preliminaries

This chapter provides foundational information for the banking chatbot system, designed to handle customer queries in the Indian financial. The system leverages Retrieval-Augmented Generation (RAG) for retrieving relevant documents and generating accurate responses, and Reinforcement Learning (RL) to optimize query-handling efficiency. The following sections define key methodologies, models, and libraries, and detail the machine learning paradigms used, including their algorithms and formulas.

The paper **"Reinforcement Learning for Optimizing RAG for Domain Chatbots" [1]** (accepted at AAAI 2024 Workshop on Synergy of RL and LLMs) proposes a novel reinforcement learning (RL) framework to improve retrieval-augmented generation (RAG) for enterprise-scale domain-specific chatbots. This paper is used as an inspiration for the entire project workflow. In the later stages, this paper has been reproduced entirely in the project. There still exist differences in the models / datasets / tools used and that indeed shows the quality of results obtained in both the cases.

#### 3.1 Definitions

This section defines all key methodologies, models, and libraries used in the banking chatbot system, specifying their roles and applications. These definitions provide a clear understanding of the technical components driving the system’s functionality.

**Large Language Models (LLMs)**

The project leverages advanced Large Language Models (LLMs), which are AI systems trained on vast text datasets to comprehend and produce human-like language, enabling sophisticated tasks such as text generation and contextual understanding. Specifically, the Groq API’s "llama3-70b-8192" model serves as the backbone for response generation within the Retrieval-Augmented Generation (RAG) system, harnessing its robust capacity to deliver context-aware, accurate answers tailored to banking queries in compliance with Indian financial regulations. Additionally, the "llama3-8b-8192" model is employed in the evaluator logic to assess response quality, chosen for its computational efficiency, which ensures rapid and reliable evaluation of the system’s outputs while maintaining alignment with regulatory standards.

**Python Libraries**

The following core python libraries have been used in the project for successful implementation.

* **FAISS - Facebook AI Similarity Search**

FAISS, a highly efficient library designed for similarity search and clustering of dense vectors, is integral to the RAG system of the conversational AI platform. Optimized for large-scale retrieval tasks, it indexes embedded instructions stored in the knowledge base ("final\_knowledge\_base.jsonl"), enabling rapid and precise document retrieval. By facilitating fast similarity searches, FAISS ensures that the system can quickly identify and retrieve relevant banking-related information, enhancing the accuracy and responsiveness of the assistant’s answers to user queries.

* **PyTorch**:  
  PyTorch, a robust deep learning framework renowned for its support of tensor computations and GPU acceleration, underpins the development and training of the BertPolicyNetwork within the Reinforcement Learning (RL) system. This framework enables the construction of neural networks that optimize context-fetching decisions, determining whether to retrieve additional information or rely on existing knowledge. By leveraging PyTorch’s flexibility and computational power, the system efficiently trains the policy model to make informed decisions, improving the assistant’s performance in handling banking queries.
* **Transformers (Platform used: HuggingFace)**

The Transformers library from Hugging Face, a cornerstone for natural language processing tasks, provides access to pre-trained models that enhance the conversational AI system’s text processing capabilities. It supplies BERT ("bert-base-uncased") and Sentence Transformer ("all-mpnet-base-v2") models, which are employed in both the RAG and RL systems for text embedding and classification. These models enable the system to generate accurate embeddings for document retrieval and process conversational states, ensuring that responses are contextually relevant and compliant with Indian banking regulations.

* **scikit-learn**:  
  scikit-learn, a versatile machine learning library, supports the evaluation of the conversational AI system’s policy models by providing tools for data analysis and performance metric computation. It calculates key metrics such as precision, recall, F1 score, and confusion matrices, which are critical for assessing the accuracy and effectiveness of the BertPolicyNetwork’s context-fetching decisions. By enabling rigorous evaluation, scikit-learn ensures that the system identifies areas for improvement, enhancing the reliability of the banking assistant’s responses.
* **matplotlib**:  
  matplotlib, a powerful plotting library, facilitates the visualization of evaluation results in the conversational AI system, offering clear insights into model performance. It generates visualizations such as confusion matrices and action count bar charts, which illustrate the policy model’s behavior in deciding between fetching additional context or responding directly. These visual representations aid developers and stakeholders in understanding the system’s effectiveness, supporting informed refinements to improve response quality and efficiency.
* **tqdm**:  
  tqdm, a Python library that adds progress bars to iterative processes, enhances the user experience during computationally intensive tasks in the conversational AI system. It provides real-time feedback on the progress of training the policy model and generating datasets for reinforcement learning. By improving transparency and interactivity, tqdm ensures that developers can monitor long-running processes effectively, streamlining the development and optimization of the banking assistant.
* **pathlib**:  
  pathlib, a library for platform-independent file system path handling, streamlines file operations within the conversational AI system. It manages tasks such as checking the existence of dataset files during the generation of reinforcement learning training data. By ensuring robust and consistent file handling across different operating systems, pathlib supports the system’s reliability, particularly when processing conversation histories and datasets stored in JSONL format.
* **json**:  
  The json library, essential for parsing and generating JSON data, supports the conversational AI system’s data management needs. It handles the processing of JSONL files, such as conversation histories ("conv\_history.jsonl" or "conversations.jsonl") and reinforcement learning datasets ("rl\_dataset\_new.jsonl"). By enabling structured data interchange, json ensures that the system can efficiently store, retrieve, and analyze interactions, maintaining compliance and facilitating debugging and training processes.
* **Hashlib (MD5)**

hashlib, specifically its MD5 functionality for generating cryptographic hash values, optimizes API interactions in the conversational AI system’s model performance evaluation. It creates unique cache keys for query-context pairs, stored in the "rag\_cache" directory, to prevent redundant API calls and ensure data integrity. This caching mechanism enhances the system’s efficiency, reducing computational overhead and supporting seamless evaluation of policy model performance.

**Loss Functions**

This part defines the loss functions used and explains their significance in the project.

**Cross-Entropy Loss**

Cross-Entropy Loss is a fundamental loss function for classification tasks, quantifying the discrepancy between predicted probabilities and true labels to guide model optimization during training. In the conversational AI system for banking queries, it is employed to train the BertPolicyNetwork within the Reinforcement Learning (RL) system, enabling accurate classification of queries into "FETCH" or "NO\_FETCH" actions based on conversational context. The loss function measures how well the model’s predicted probabilities align with the true action labels, optimizing the network to make informed context-fetching decisions that enhance response accuracy and compliance with Indian financial regulations. The

Formula for this loss function is as follows:

where is the true label (0 for "NO\_FETCH", 1 for "FETCH"), and is the predicted probability of the "FETCH" action for the i-th sample.

**InfoNCE loss**

InfoNCE Loss, or Noise-Contrastive Estimation Loss, is a contrastive loss function used to train embedding models by maximizing the similarity between positive pairs while minimizing similarity with negative pairs, thereby improving representation learning. In the project, InfoNCE Loss is utilized to fine-tune the embedding model "e2-base-v5," which was pre-trained on English and Hinglish paraphrases of original FAQ content. This fine-tuning enhances the model’s ability to generate high-quality embeddings for the knowledge base, enabling more accurate document retrieval in the Retrieval-Augmented Generation (RAG) system by capturing semantic similarities relevant to the Indian banking queries. The formula for this loss function is:

A mathematical equations and formulas

AI-generated content may be incorrect.

where zi and zj are embeddings of a positive pair, zk includes negative samples, sim denotes a similarity function (e.g., cosine similarity), τ is a temperature parameter, and K is the number of negative samples.

**Triplet Loss**

Triplet Loss is a loss function designed to optimize embedding models by ensuring that the distance between an anchor and a positive example is smaller than the distance between the anchor and a negative example, thus improving the quality of learned embeddings. In the conversational AI system, it is used to fine-tune the same "e2-base-v5" embedding model, pre-trained on English and Hinglish paraphrases of FAQ content, to enhance its capability to produce embeddings that effectively distinguish relevant from irrelevant documents in the RAG system. This fine-tuning supports precise document retrieval for banking queries, ensuring responses are contextually accurate and compliant with regulations. The formula for Triplet loss is:



where ( a ) is the anchor embedding, ( p ) is the positive embedding, ( n ) is the negative embedding, ( d ) is a distance metric (e.g., Euclidean distance), and ( margin ) is a hyperparameter enforcing separation between positive and negative pairs.

**Monte Carlo Dropout Method**

The Monte Carlo Dropout Method is a technique that introduces randomness during inference by enabling dropout layers in a neural network, allowing multiple forward passes to estimate uncertainty and improve decision robustness. In the project, it is applied to finalize the action taken by the BertPolicyNetwork in the RL system, where it computes ten probability or confidence scores for the "FETCH" or "NO\_FETCH" actions. By averaging these scores, the method ensures a stable and reliable final action, enhancing the system’s ability to make consistent context-fetching decisions for banking queries while mitigating overfitting risks. The method does not involve a specific loss function but relies on the following

averaging process for probabilities:

A math equation with numbers and symbols

AI-generated content may be incorrect.

where pt is the predicted probability for an action in the tth forward pass, T = 10 is the number of passes and pfinal is the average of the predicted probabilities.

**Cross-Encoders**:  
Cross-Encoders are advanced models designed to enhance ranking accuracy by simultaneously encoding query-document pairs to score their relevance, making them ideal for refining search results. In the conversational AI system, the CrossEncoder ("cross-encoder/ms-marco-MiniLM-L-6-v2") is employed within the Retrieval-Augmented Generation (RAG) system to rerank retrieved contexts, ensuring that the most relevant banking-related documents are prioritized for generating responses. This process, critical as of 02:31 PM IST on July 25, 2025, strengthens the system’s ability to deliver accurate and contextually appropriate answers aligned with Indian financial regulations.

**Embedding Models:**

Embedding models are specialized systems that transform text into dense vector representations, facilitating tasks such as similarity search by capturing semantic relationships. Within the project, the Sentence Transformer ("all-mpnet-base-v2") plays a key role by generating embeddings for instructions stored in the knowledge base ("final\_knowledge\_base.jsonl"), enabling efficient retrieval of relevant documents in the RAG system. Additionally, the "e2-base-v5" model, pre-trained on English and Hinglish paraphrases of original FAQ content, is fine-tuned using InfoNCE and Triplet Loss to further enhance its embedding quality, supporting precise document matching for banking queries as of the current date and time.

**Models used:**

The conversational AI system leverages a diverse set of models to handle various tasks, each tailored to specific functionalities.

The BertPolicyNetwork, a custom BERT-based neural network designed for classification, predicts whether to "FETCH" or "NO\_FETCH" actions in the Reinforcement Learning (RL) system, basing its decisions on query and conversation history to optimize response strategies. Its foundation, the BERT ("bert-base-uncased") pre-trained transformer model, provides deep contextual text understanding,

The Sentence Transformer ("all-mpnet-base-v2") generates sentence-level embeddings for instructions in the knowledge base, enhancing retrieval accuracy in the RAG system.

The CrossEncoder ("cross-encoder/ms-marco-MiniLM-L-6-v2") ranks query-document pairs to improve context relevance during retrieval.

For response generation, the Groq API’s "llama3-70b-8192" high-capacity Large Language Model (LLM) produces complex, context-aware responses, while the "llama3-8b-8192" smaller, efficient LLM evaluates response quality in the evaluator logic, ensuring computational efficiency.

Additionally, the "e2-base-v5" model, fine-tuned on FAQ paraphrases, contributes to embedding refinement.

Lastly, the newly integrated "meta-llama/llama-4-scout-17b-16e-instruct" model, a specialized instruction-tuned LLM, supports advanced response generation and instruction-following tasks.

**3.2 Machine Learning Paradigms**

This section details the machine learning paradigms used in the banking chatbot system: Retrieval-Augmented Generation (RAG) and Reinforcement Learning (RL). It explains their applications, algorithms, and formulas, ensuring a comprehensive understanding of their roles.

**Retrieval-Augmented Generation (RAG)**

Retrieval-Augmented Generation (RAG) is a methodology that integrates document retrieval with generative models to produce context-aware responses, making it ideal for tasks requiring external knowledge.

By combining a retriever to fetch relevant documents and a generator to produce answers, RAG enhances the ability to provide informed and accurate responses. RAG improves **factual accuracy**, helps LLMs (Large Language Models)overcome their **knowledge cutoff** or **lack of real-time knowledge** and provides more **grounded** and **context-aware** answers.

In the project, RAG is employed to handle banking queries by retrieving relevant documents from a knowledge base, such as "final\_knowledge\_base.jsonl," using FAISS and a fine-tuned embedding model, using infoNCE loss and triplet loss. The retrieved documents are then used by the Groq API ("llama3-70b-8192") to generate responses.

A reinforcement learning (RL) system determines whether additional context retrieval is necessary.

The RAG also uses a reranker, which employs a CrossEncoder ("cross-encoder/ms-marco-MiniLM-L-6-v2") to rank retrieved documents for relevance.

A black and white math equation

AI-generated content may be incorrect.

Above mentioned formula is cosine similarity which is used during retrieval process. Here, ( q ) represents the query embedding, and ( d ) represents the document embedding.

**Reinforcement Learning (RL)**

Reinforcement Learning (RL) is a framework where an agent learns optimal actions by interacting with an environment, receiving rewards or penalties to maximize cumulative reward, making it well-suited for decision-making tasks.

In the project, RL is used to train a BertPolicyNetwork to decide between two actions: "FETCH," which retrieves additional context, or "NO\_FETCH," which relies on general knowledge, optimizing both efficiency and accuracy for each query. The key components of the RL system include the agent, implemented as the BertPolicyNetwork, which predicts actions based on the query and conversation history. The environment consists of the conversational context, including the query, history, and knowledge base. The rewards are assigned as follows: 0.5 for FETCH, 2.0 for NO\_FETCH with a good response, and -0.5 for NO\_FETCH with a bad response. The model is trained over 100 epochs using a dataset of past interactions, where actions are randomly selected, responses are evaluated, and rewards are assigned to guide learning.

The RL system employs the policy gradient method to update policy parameters and maximize expected cumulative reward. Policy Gradient formula:

A black and white math symbol

AI-generated content may be incorrect.

represents the policy probability of taking action at at a given state​ st . At is the advantage (often approximated by the reward or a baseline-adjusted reward), and the expectation E is taken over the trajectory of actions. The use of the vertical bar | is standard for conditional probability, and the formula is correctly structured to describe the policy gradient method used to update the parameters θ of the policy to maximize the expected cumulative reward.

## Chapter 4: Proposed Work

#### 4.1 Methodology

The methodology employs an integrated yet modular framework that combines **Retrieval-Augmented Generation (RAG)** with **Reinforcement Learning (RL)** to develop a robust, adaptive banking chatbot. It is structured into five interconnected components forming a complete pipeline from system design to policy optimization. The approach ensures accurate, regulation-compliant responses while continuously improving decision-making efficiency.

**1. Conversational AI System (RAG Framework)**

**a) Purpose**

To create a banking assistant capable of accurately answering user queries by leveraging a knowledge base built on Indian banking regulations and practices.

**b) Key Components**

* **Knowledge Base:**  
  Stored in *“final\_knowledge\_base.jsonl”*, it contains banking-specific FAQs and compliance-related instructions. Each entry is embedded using a **fine-tuned “e2-base-v5” Sentence Transformer model**, optimized with *InfoNCE* and *triplet loss* on 1665 English–Hinglish paraphrased FAQs. This produces semantically rich embeddings for highly accurate retrieval via **FAISS**.
* **Query Classification:**  
  A **BERT-based policy model (“BertPolicyNetwork”)**, built on *“bert-base-uncased”*, classifies each query as either **“FETCH”** (context needed) or **“NO\_FETCH”** (direct response). A confidence threshold of 0.75 determines the final action. The model is loaded from its trained checkpoint (e.g., *“policy\_rl\_20250705\_111202.pt”*) and runs in evaluation mode.
* **Context Retrieval:**  
  For “FETCH” queries, the system retrieves the **top 3,000** relevant instructions from the FAISS index using embeddings generated by the fine-tuned retriever.  
  *(Unlike earlier implementations, no CrossEncoder is used for reranking — the fine-tuned retriever alone ensures strong relevance precision.)*  
  The top 5 most relevant results are selected for response generation.
* **Response Generation:**  
  The **Google API (gemini-2.0-flash-lite)** model generates natural, compliant responses. The prompt includes:
  + the retrieved context (if any),
  + the last four conversational turns for continuity, and
  + the user’s query.
* **Logging:**  
  All interactions — queries, responses, session data, and timestamps — are stored in *“conv\_history.jsonl”* to enable compliance tracking and serve as training data for RL.

**c) Workflow**

A user query (e.g., “What are the current home loan interest rates?”) is classified as “FETCH” or “NO\_FETCH.”  
If “FETCH,” the retriever fetches context via FAISS.  
Groq generates the final response based on the context, history, and query, which is then logged for future evaluation and RL.

**2. Response Evaluation (Evaluator Logic)**

**a) Purpose**

To evaluate response quality and provide structured feedback for reinforcement learning.

**b) Key Components**

* **Evaluation Criteria:**  
  Each response is labeled **“GOOD”** (accurate, contextual, compliant) or **“BAD”** (irrelevant, fabricated, or non-compliant).  
  “NO\_FETCH” responses are held to a stricter accuracy standard.
* **Reward System:**

| **Type** | **Reward** | **Description** |
| --- | --- | --- |
| GOOD “NO\_FETCH” | +2.0 | Highly efficient and accurate |
| GOOD “FETCH” | +0.5 | Correct but less efficient |
| BAD “NO\_FETCH” | −0.5 | Penalizes wrong direct responses |

* **Evaluation Process:**  
  Uses the **Groq API (LLaMA3-70B-8192)** model for cost-effective, consistent evaluations through standardized prompts.

**c) Workflow**

Each generated response is evaluated, assigned a label and reward, and logged. These logs directly feed into the RL dataset for policy model training.

**3. Dataset Generation for Reinforcement Learning**

**a) Purpose**

To simulate interactions and build a dataset containing state–action–reward tuples for RL training.

**b) Key Components**

* **Data Source:**  
  Extracted from *“conv\_history.jsonl”*, containing real chat sessions.  
  (The history includes six chat sessions and 182 user queries.)
* **Action Sampling:**  
  For each query, the system explores both “FETCH” and “NO\_FETCH” actions, selected randomly with equal probability to encourage exploration.
* **Response & Evaluation:**  
  Responses generated for each action are evaluated by the evaluator logic, and corresponding rewards are assigned.
* **Output:**  
  Each tuple — (state, action, reward) — is stored in *“rl\_dataset\_new.jsonl”* to train the policy model.

**c) Workflow**

This component simulates user interactions under different actions to ensure diverse, high-quality RL training data.

**4. Policy Network Training with Reinforcement Learning**

**a) Purpose**

To train the BERT-based policy model to make optimal “FETCH” vs. “NO\_FETCH” decisions.

**b) Key Components**

* **Model Architecture:**  
  Built on *“bert-base-uncased”* with dropout (0.3) to prevent overfitting and a classification head predicting probabilities for the two actions.
* **Training Data:**  
  The model is trained using **randomly shuffled query-response pairs** from the conversation history, forming (state, action, reward) tuples.
* **Training Process:**  
  Runs for **100 epochs** with a batch size of 32, using the **AdamW optimizer** (learning rate = 2e-5) and **entropy regularization (λ = 0.3)** to maintain exploration.
* **Output:**  
  The optimized policy network is saved as *“policy\_rl\_{timestamp}.pt”* for deployment.

**c) Workflow**

The policy learns from past rewards to minimize unnecessary retrieval while maintaining high response quality, balancing efficiency with accuracy.

* **RAG (Retrieval-Augmented Generation) Methodology**

1. **Purpose**: To develop a banking assistant that delivers accurate and compliant responses using a knowledge base.
2. **Key Components**:

- Knowledge Base: A collection of banking documents (e.g., stored in "final\_knowledge\_base.jsonl") containing detailed information on Indian banking and financial regulations, such as loan policies, account management.

- Tools:

* FAISS: Enables efficient similarity searches by indexing document embeddings.
* Sentence-transformers: Generates embeddings for queries and documents, facilitating semantic search.
* Groq API: Powers response generation with the "llama3-70b-8192" model, known for its ability to handle complex queries.

1. **Main Functions**:

- retrieve\_documents: Searches the knowledge base for documents similar to the user query, using a

similarity threshold (e.g., 0.5). Returns the top 5 documents with metadata like similarity scores,

categories (e.g., "Loans", "Savings Accounts"), and ranks.

- format\_context: Summarizes retrieved documents into a concise context string (limited to 2000 characters) to provide relevant information to the language model without overwhelming it.

- generate\_with\_groq: Generates responses using the Groq API, incorporating the formatted context, conversation history (last 4 turns), and specific prompts to ensure professionalism.

- process\_query: Orchestrates the RAG pipeline by coordinating document retrieval, context formatting, and response generation. Returns the final answer, retrieval details, and document list for transparency.

- log\_conversation: Logs all interactions, including queries, responses, and retrieval metadata, to "conversations.jsonl" with timestamps and unique session IDs for compliance and analysis.

1. **Workflow**:

- A user query (e.g., "What are the eligibility criteria for a personal loan?") triggers "process\_query".

- "retrieve\_documents" fetches relevant documents from the knowledge base using FAISS,

prioritizing those with high similarity scores. If no documents could be fetched, it prints the relevant message.

- "format\_context" summarizes these documents into a concise context string.

- "generate\_with\_groq" produces a professional, compliant response using the context, history, and

prompts.

- "log\_conversation" records the interaction for auditing and future training.

1. **Error Handling**: The system includes robust error handling for API failures, ensuring reliability in real-world deployment such as API key rotations, bucket limits so that tokens used do not exceed the actual daily limit set by the AI service platform, here [GroqCloud](https://console.groq.com/home).

* **RL + Policy + Evaluator Methodology**

1. **Purpose**: To optimize decision-making by determining when to fetch context ("FETCH") or respond directly ("NO\_FETCH"), minimizing computational costs while maintaining accuracy.
2. **Key Components**:

- Reinforcement Learning Framework:

* Uses a reward-based system to train the chatbot: +2.0 for "GOOD" "NO\_FETCH" (efficient and accurate), +0.5 for "FETCH" (necessary but less efficient), and -0.5 for "BAD" "NO\_FETCH" (penalizing incorrect direct responses).
* Aims to reduce unnecessary context fetches, improving response time and cost-efficiency.

- Evaluator AI:

* + Rates responses as "Good" (accurate, relevant, compliant) or "Bad" (inaccurate, irrelevant) using an AI model guided by detailed evaluation prompts.
  + Ensures consistent and fair evaluation of response quality.

1. **Main Functions:**

- evaluate\_response: Assesses response quality using an AI model, typically the Groq API with a smaller model (e.g., "llama3-8b-8192") for cost efficiency.

- get\_reward\_from\_rating: Converts evaluator ratings into numerical rewards based on the reward system.

- count\_tokens: Tracks API token usage to manage costs and optimize resource allocation.

- create\_rl\_training\_dataset: Simulates conversations to generate training data, including states

(query and context), actions ("FETCH" or "NO\_FETCH"), and rewards.

- train\_policy\_network: Trains a BERT-based policy network ("PolicyNetwork") using the generated dataset to predict optimal actions.

- test\_policy\_network: Evaluates the policy network’s performance on test data, using metrics like accuracy and cumulative rewards.

- **process\_conversation\_log\_for\_rl**: Extracts training data from past logs ("conversations.jsonl") to ensure the model learns from real-world interactions.

- **format\_history**: Prepares conversation history for training, ensuring the policy network considers contextual information.

1. **Policy Network**:

- A BERT-based model ("bert-base-uncased") that processes the current query and conversation history to predict "FETCH" or "NO\_FETCH".

- Trained using RL to maximize cumulative rewards, balancing accuracy and efficiency.

1. **Workflow**:

    - The policy network assesses the query and history to decide "FETCH" or "NO\_FETCH".

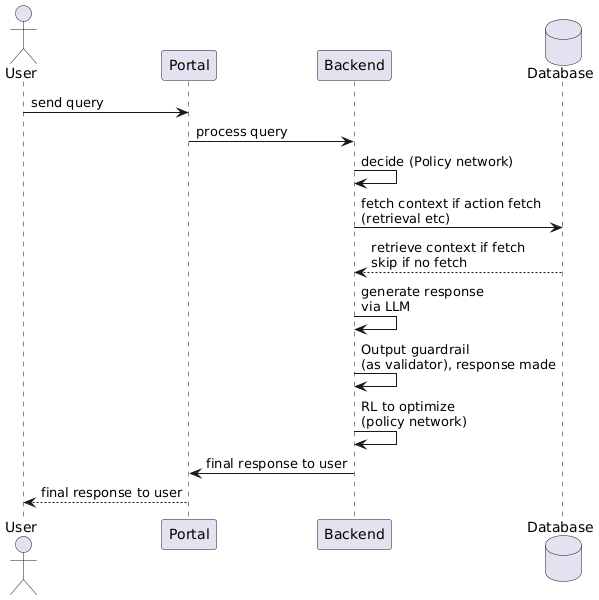
- If "FETCH", the RAG system retrieves and formats context; otherwise, a direct response is generated.

- The response is evaluated, a reward is assigned, and the policy network is updated through training.

- The "run\_complete\_rl\_pipeline" function orchestrates this entire process, ensuring that the continuous improvement of the policy model takes place.

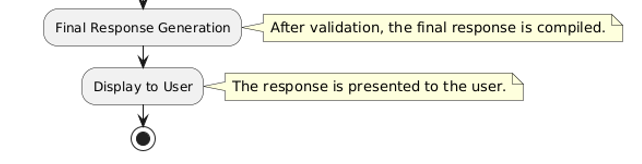
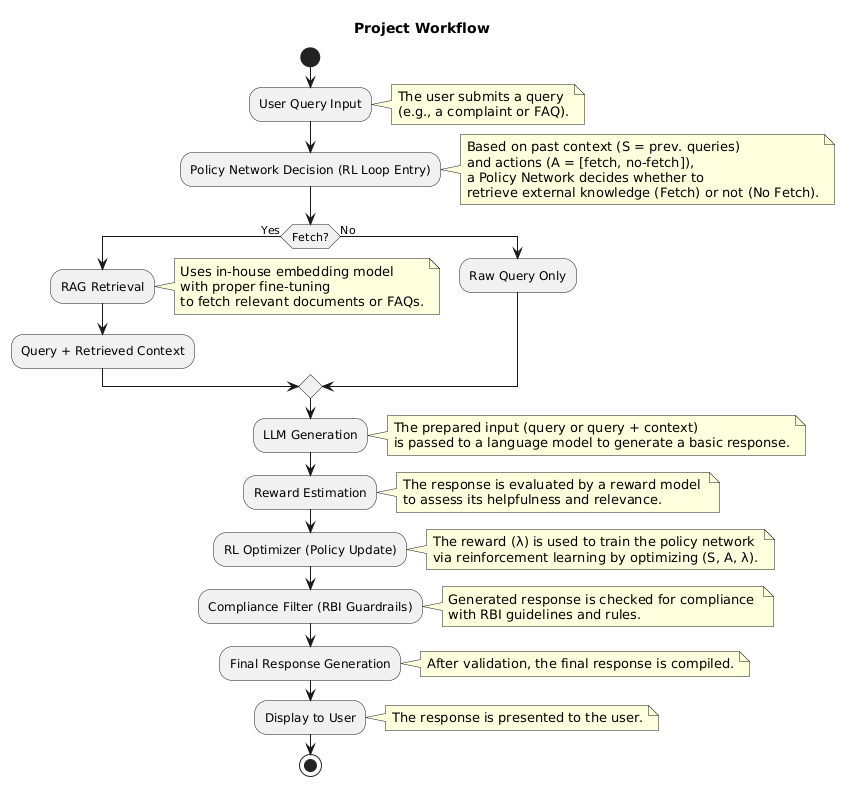
#### 4.2 Diagrams

**4.2.1 Sequence Diagram**

****

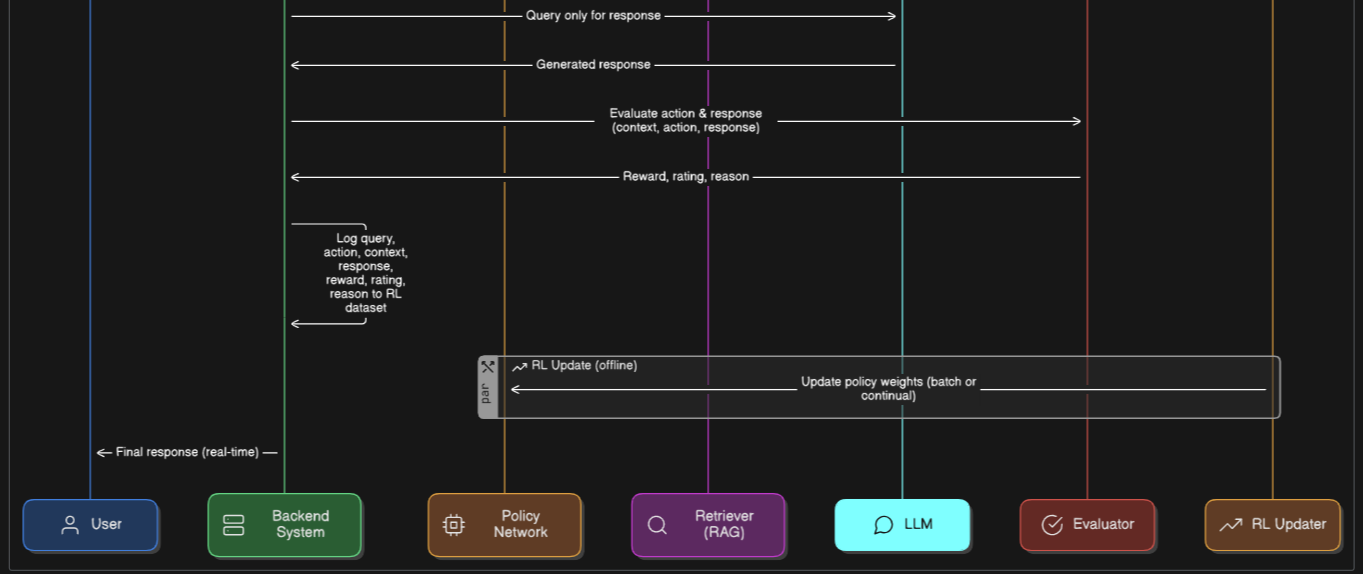
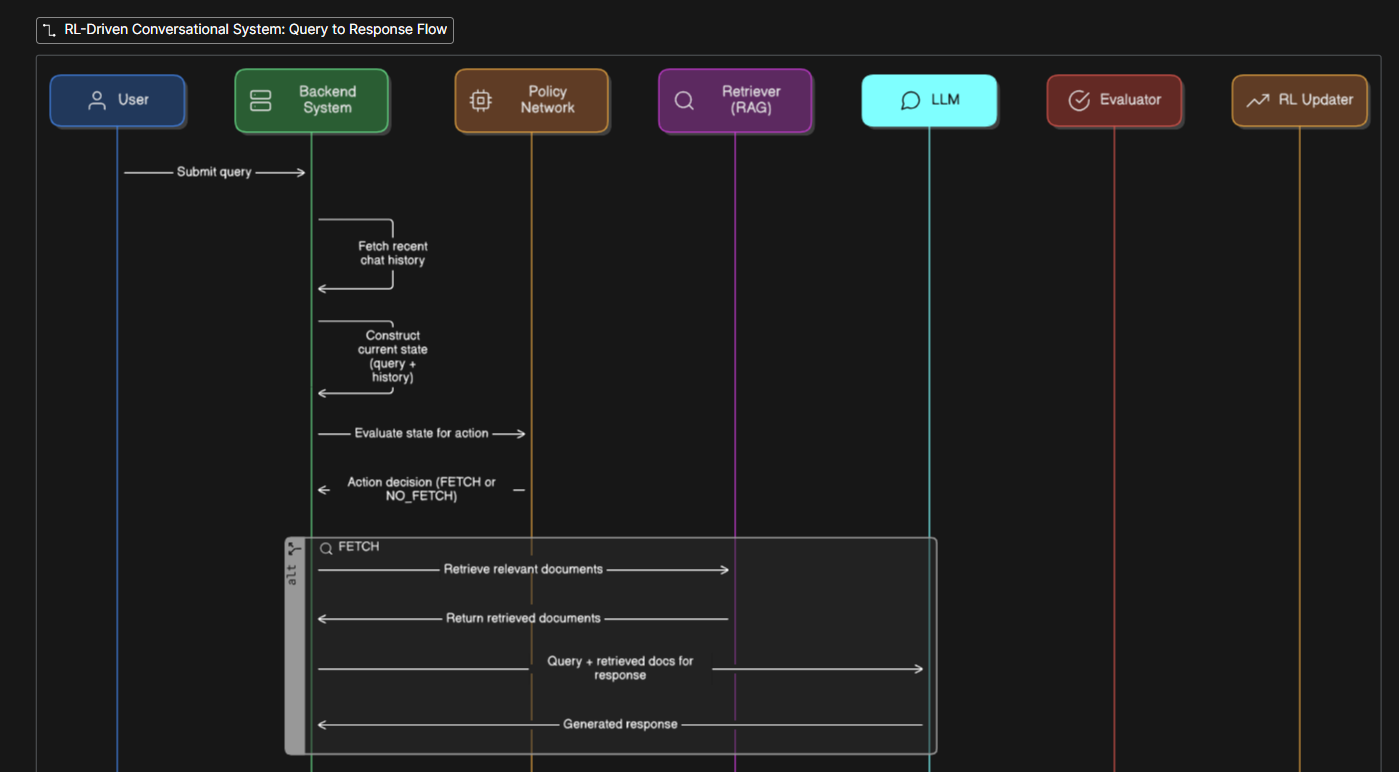
**Credit:** [**Plant UML**](https://plantuml.com/)

**4.2.2 Architecture Diagram**



**Credit:** [**Plant UML**](https://plantuml.com/)

**4.2.3 Backend Flow diagram**



**Credit:** [**Eraser IO**](https://app.eraser.io/new)

## Chapter 5: Experiments

#### 5.1 Dataset preparation

##### **5.1.1 Final Knowledge Base**

A comprehensive dataset was developed and used in the form of the final knowledge for the RAG pipeline. This dataset was prepared by using 3 different datasets out of which, 2 are open source datasets and 1 is manually created dataset.

**The open-source datasets are:**

1. **Bitext Dataset** **[14]:** This dataset is directly taken from hugging face website. This synthetic dataset is built for fine-tuning LLMs (like GPT, Mistral, OpenELM) in the **Retail Banking** domain. It supports **intent detection**, with **26 intents** across **9 categories**, **25,545 Q&A pairs**, and includes **1,224 entities** and **12 NLG tag types**. It’s part of a two-step domain adaptation process—train on this dataset, then fine-tune further with your own data.

A screenshot of a computer

AI-generated content may be incorrect.This screenshot is of the already pre-processed and clean dataset.

1. A screenshot of a computer

   AI-generated content may be incorrect.**Conversation Dataset** **[15]:** RetailBanking-Conversations is a synthetic dataset built with wizardSdata for training/evaluating LLMs in retail banking. It includes **320 conversations** across **160 unique financial profiles** and **10 key topics**, simulating realistic advisor-client dialogues. **Conversations average 4–8 turns**, all in **English**.

This screenshot is of the already pre-processed and clean dataset.

**Manually created dataset:**

1. **Bank FAQs dataset:** This dataset was created by **manually selecting** FAQs from real bank websites. These FAQs cover the **main 4 intents** which are used in the final knowledge base: **ATM, ACCOUNT, CARD, LOAN**. The size of this dataset is **1016** Real bank FAQs. **Banks** chosen are: SBI, icici, hdfc, yes bank and axis bank.

**This is the original form of the dataset in the form of .txt file, later on converted to proper json file.**

A screenshot of a computer screen

AI-generated content may be incorrect.

This is purely done to improve the dataset quality in my project. The **final knowledge base** consists of **15 categories, 22 intents** and a total of **19352** question-answering pairs. Out of these, the following **4 categories (ATM, CARD, LOAN, ACCOUNT)** are considered in this project which have total of **16911** question-answering pairs.

##### **5.1.2 Policy Network Pre-Training Dataset**

This dataset was created by using labelled queries. **2 types of queries** were used. **FETCH (labelled as 1)** and **NO\_FETCH (labelled as 0)** were used. **Equal split** between the 2 types of the mentioned queries. The **total value** of this dataset comes to be **10083 + 10083 = 20166.**

##### **5.1.3 Retriever Dataset**

This dataset was created by using 1665 unique FAQs from the final knowledge base. Paraphrases were then formed for these unique FAQs in both English and Hinglish. 4 English paraphrases per FAQ, 2 Hinglish paraphrases per FAQs and also include the original FAQ in the corpus to give context. So, the total number of values in this dataset comes to be around 11655. This dataset was used in the training of the Retriever model in the project.

#### 5.2 Experiments:

##### **5.2.1 Policy Model**

**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

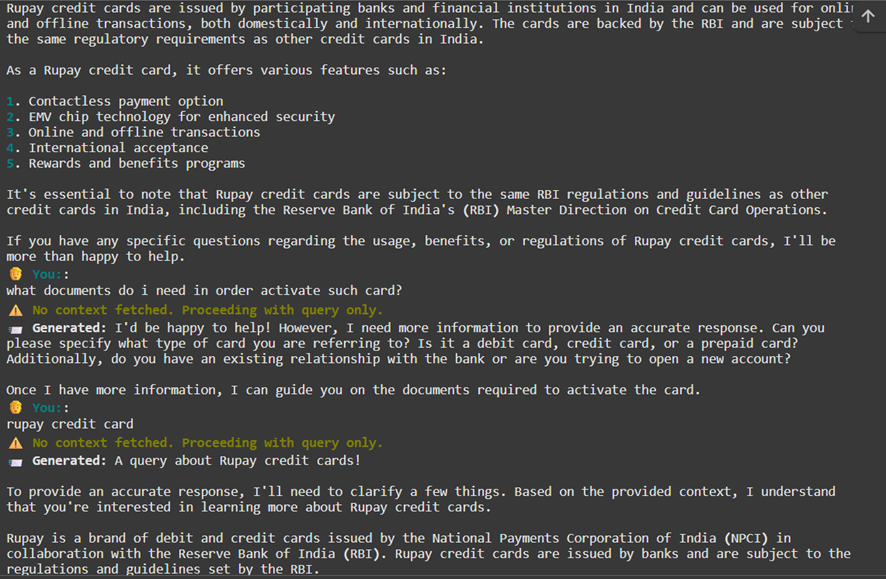
AI-generated content may be incorrect.**

* The following images show the testing of Policy network in the project.
* The first image shows the manual query testing of the Policy Model with its corresponding results.
* The second image, on the other hand, shows the training as well as the entering of the manual queries in the console itself.
* The policy model gives the decision as FETCH or NO\_FETCH depending on the query.

##### **5.2.2 RAG**



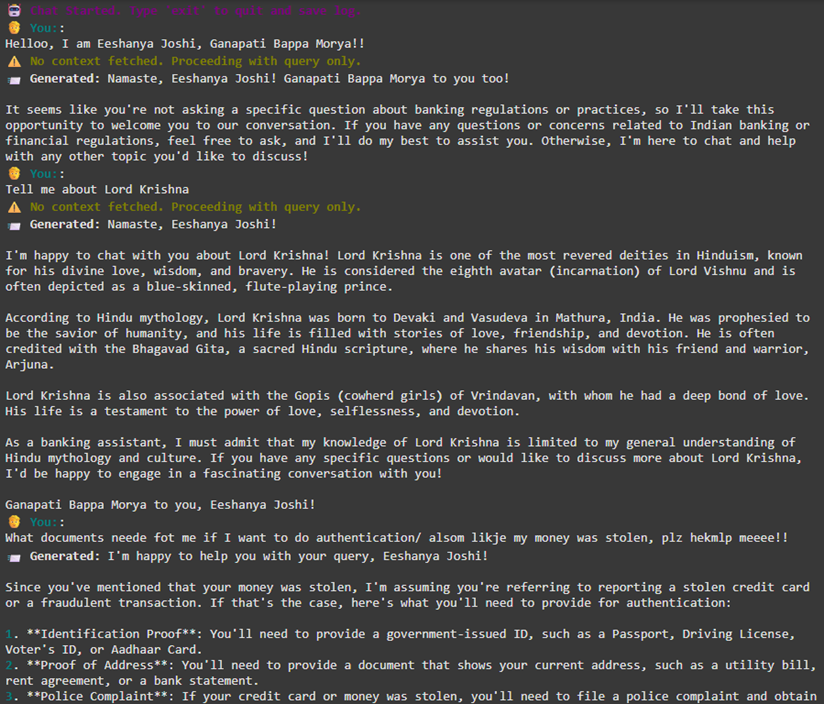
**Figure: Chat session with RAG system**



**Figure: Continued Chat session with the RAG system.**

* These images show the simulation of the RAG pipeline of this project.
* The first image shows a query for which a FETCH action is being performed, and the bot is responding to the query accordingly.
* The second image shows a query for which a NO\_FETCH action is being performed, and the bot first prints a small statement which says “No context fetched. Proceeding with query only.”
* This means that the bot did not retrieve any context from the knowledge base and directly gave the response.

##### **5.2.3 RL + RAG + Policy updates**



**Figure: Chat session going on with the bot, responses are improved.**

A screenshot of a computer screen

AI-generated content may be incorrect.

**Figure: Continued chat session with the system, improved responses.**

* These images show the simulation of the RAG pipeline with RL updation of the policy model in this project.
* Both the images show NO\_FETCH action performed on queries. Though it answers the queries gracefully, it also says that as a banking assistant, its knowledge is limited to the respective domain and kindly ask queries related to the banking domain only.
* This means that the bot did not retrieve any context from the knowledge base and directly gave the response.

#### 5.3 Experiments: here we have reproduced the [1] paper.

##### **5.3.1 Policy Training and RL update**

**A screenshot of a computer program

AI-generated content may be incorrect.Figure: Failed outcome, Execution terminated due to code error and rate limitations**

**A screenshot of a computer

AI-generated content may be incorrect.Figure: Outcome successfully completed**

**Figure: RL Training of Policy Model**

**A black rectangular object with white text

AI-generated content may be incorrect.A black rectangular object with white text

AI-generated content may be incorrect.**

##### **5.3.2 Retriever (3 types)**

**A screenshot of a computer

AI-generated content may be incorrect.Figure: Paraphrase generation**

**Paraphrases have been generated using the groq model: llama3-70b-8192 model using the following prompt: “You are given 1665 FAQs. You have to generate random paraphrases which resemble the original query. 4 English paraphrases have to be taken and 2 Hinglish paraphrases have to be taken. Do not hallucinate and generate garbage response, carefully analyze the given input query and generate relevant paraphrase for the same.”**

**Figure: Paraphrase generation complete, dataset statistics**

**A screenshot of a computer

AI-generated content may be incorrect.**

**experimentations have been performed in 3 different setups:**

**1. Neither InfoNCE loss norTriplet loss**

**2. Only InfoNCE loss**

**3. Both InfoNCE loss and Triplet loss**

**A screenshot of a computer

AI-generated content may be incorrect.Type-1 Without infoNCE and triplet loss**

**Figure: dataset statistics**

**Figure: started training**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.Type-2 With infoNCE only**

**Figure: dataset statistics**

**Figure: training started.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.Type-3 With infoNCE and triplet loss**

**Figure: Dataset information**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Figure: started training**

##### **5.3.3 RAG Pipeline**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Figure: Chat interaction.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Figure: Continued chat interaction.**

## Chapter 6: Results and Conclusions

This chapter presents the results obtained in the project. The main objective of this chapter is to show the number of tokens used and which type of result uses the least number of tokens by providing a decent accuracy.

The main goal of this project is to observe the token cost and accuracy of the models. Then it can be suggested to choose the appropriate form of the model as per requirements. In this project, there are four experimental setups considered, as follows,

**- Always fetch:** This is the version where there is no role of SimThr and technically, policy network as well. This version always uses RAG pipeline no matter what. Hence, as such no requirement of policy network but still, in this case, there is a need to test using both the versions of the policy model and that exactly done so.

**- SimThr+Policy:** This is the final version and the main version which is currently being used for the project. It uses both SimThr and the policy network to determine when to use the RAG pipeline and when not to use it. This version definitely uses both policy models, base and the RL version as well.

**- SimThr only:** (SimThr is the similarity Threshold value. This means that if the similarity score exceeds that value then the system will directly use RAG pipeline. Policy network is never used here, which also means there will be no “NO\_FETCH” actions and as a result, RL is also never used in this variation). Similarity score is basically used to determine when to fetch from the RAG pipeline, used by cross encoder and faiss embeddings.

**- Policy only:** This is the version in which only the policy network is used as the deciding factor. There is no SimThr. This version uses both policy models, base and the RL version as well.

#### 6.1 Results (original)

**WITH RL:**

|  |  |  |
| --- | --- | --- |
| Type | Token Cost | Accuracy |
| Always Fetch | 137482 | 48% |
| Similarity Threshold + Policy Model | 151557.5 | 60% |
| Similarity Threshold only | 57876 | 50% |
| Policy Model only | 69873 | 53.04% |

**WITHOUT RL:**

**A screenshot of a document

AI-generated content may be incorrect.**

#### 6.2 Results (improved)

Following are the brief results obtained by the 4 types of evaluation methods.

**WITH RL**

|  |  |  |
| --- | --- | --- |
| Type | Token Cost | Accuracy |
| Always Fetch | 149064 | 73% |
| Similarity Threshold + Policy Model | 131136 | 92% |
| Similarity Threshold only | 55797 | 80% |
| Policy Model only | 119269 | 76% |

**WITHOUT RL**

|  |  |  |
| --- | --- | --- |
| Type | Token Cost | Accuracy |
| Always Fetch | 150064 | 63% |
| Similarity Threshold + Policy Model | 149186 | 78% |
| Similarity Threshold only | 140755 | 70% |
| Policy Model only | 85685 | 60% |

**Two Experimental Versions were Tested:**

We conducted evaluations across **two main versions** of our Retrieval-Augmented Generation (RAG) system—each tested in both **baseline** (without RL) and **RL-optimized** variants using our custom BERT-based policy network.

**Accuracy Remained Accurate:**

The **accuracy levels ranged from 43% to a maximum of 92%**, which is of high variance compared to the near-perfect (~100%) accuracies reported in some research papers. This gap is largely due to infrastructural and resource-related constraints in our environment.

**High Token Costs and Latency**

Due to reliance on external LLM APIs (Groq) for generation and evaluation, **token costs were high**, especially during RL rollouts and fine-tuning experiments. This limited the scale and depth of experimentation.

**Limited access to proprietary models and resources**

Unlike the research paper **[1]** that utilize **closed-source models or proprietary retrievers**, our system was built entirely using **open-source components** (e.g., FAISS, sentence-transformers), often lacking optimization for specific domains like banking or finance.

**RL Implementation Faced Training bottlenecks**

Though RL logic was successfully implemented for policy improvement (using offline reward logs), **frequent updates were computationally expensive**, and **evaluation batches had to be small**, reducing the stability and convergence of the policy over time.

**Gap between Theory and Real world development**

In contrast to theoretical setups in research (often using cleaner data, high compute, and proprietary access), our project focused on **practical feasibility with limited budget and compute**—which highlights a more realistic perspective on deploying RL-based RAG in enterprise chatbot settings.

## Chapter 7: References

[1] <https://arxiv.org/pdf/2401.06800>

[2] <https://chauff.github.io/documents/publications/ICTIR2021-Zhu.pdf>

[3] <https://ojs.aaai.org/index.php/AAAI/article/view/34743>

[4] <https://github.com/subrata-samanta/RL-Self-Improving-RAG>

[5] <https://github.com/ren258/ARENA>

[6] <https://github.com/NirDiamant/RAG_Techniques>

[7] <https://github.com/cse-amarjeet/Financial-RAG-From-Scratch>

[8] <https://github.com/solidx00/Bridge_the_GAP>

[9] <https://website.rbi.org.in/web/rbi/-/notifications/strengthening-of-grievance-redress-mechanism-in-banks12017>

[10] [https://website.rbi.org.in/web/rbi/-/notifications/master-direction-on-information-technology-governance-risk controls-and-assurance-practices](https://website.rbi.org.in/web/rbi/-/notifications/master-direction-on-information-technology-governance-risk%20controls-and-assurance-practices)

[11] <https://website.rbi.org.in/web/rbi/-/notifications/cyber-security-framework-in-banks-10435>

[12] <https://website.rbi.org.in/web/rbi/-/notifications/master-circular-on-customer-service-in-banks-9862>

[13] <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf>

[14] <https://huggingface.co/datasets/bitext/Bitext-retail-banking-llm-chatbot-training-dataset>

[15] <https://huggingface.co/datasets/oopere/RetailBanking-Conversations>

[16] <https://cookbook.openai.com/examples/how_to_use_guardrails>