Enhancing User Experience through an Information Retrieval Chatbot

Lakshmi Bhavana Thalapaneni (G01452929 - ltalapan@gmu.edu)
Aradhya Jain (G01462086 - ajain30@gmu.edu)
Akhila Palempati (G01448123 - apalempa@gmu.edu)

Executive Summary

In this project, we present a comprehensive Information Retrieval (IR) Chatbot that merges topic-driven query answering with dynamic, open-domain conversation. By leveraging a large, multi-topic dataset scraped from Wikipedia, employing topic classification, semantic retrieval, and a carefully tuned summarization pipeline, we aim to provide concise, contextually relevant, and human-like responses. The system seamlessly switches between providing factual answers and engaging in chit-chat when queries fall outside predefined domains. This approach required extensive hyperparameter tuning, model experimentation, and careful balancing of resource constraints against performance goals. Figure 1 illustrates the end-to-end flow of queries through classification, retrieval, and summarization stages, culminating in a user-friendly and responsive information access experience. We try to develop own idea which involves our own custom dataset, model and results. It dwells into deep knowledge of how modern LLM's work.

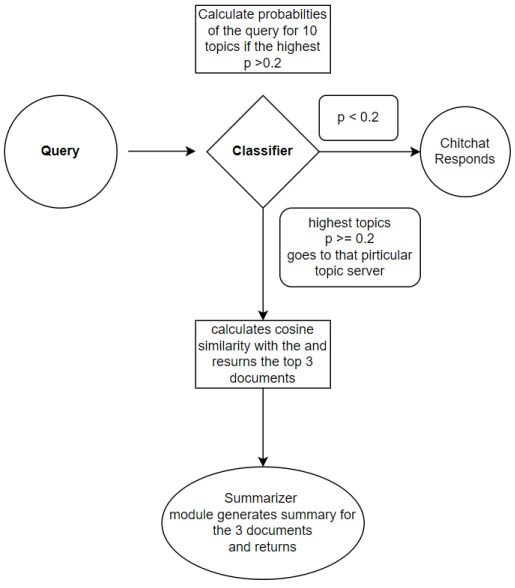

Fig. 1. System Flow Diagram: The user query is classified into a topic or deemed off-topic. If on-topic, relevant documents are retrieved and summarized. Otherwise, a chit-chat model handles the response.

Key Components and Methodologies:

- **Wikipedia Scraper:** Gathered 50,000+ unique articles across ten topics using BFS and de-duplication for high-quality datasets.
- **Chit-chat Module:** Enhanced with Blender Bot for natural conversations; outperformed DialoGPT; resource constraints limited further training.
- **Topic Classifier:** Used Naive Bayes, Logistic Regression, and a custom BERT-based model for accurate, multi-topic query classification; combined results for improved reliability.
- **Wiki Q/A Bot:** Leveraged document embeddings and cosine similarity for top-k retrieval; utilized Hugging Face for efficient summarization with optimized parameters.
- **UI and Visualization:** Web-based platform with dynamic visualizations and analytics to enhance user interaction and insights.

Key Achievements:

- Built a large-scale Wikipedia dataset for diverse topics.
- Integrated advanced chit-chat and topic classification.
- Deployed efficient summarization for real-time responses.
- Developed an intuitive, visually appealing web interface.

Recommendations:

- Scale resources for advanced training and multilingual support.
- Enhance visualization with granular analytics and behavior tracking.
- Deployed efficient summarization for real-time responses.
- Employ deep learning for improved topic classification accuracy.

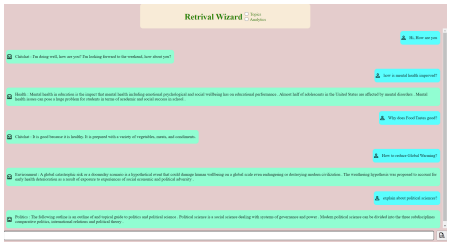

Fig. 2. Chat Interface (Web UI 1)

***Abstract*—We describe an Information Retrieval Chatbot designed to address both factual, topic-specific questions and open-domain social prompts. Our system integrates a broad Wikipedia-derived corpus, topic classification, semantic retrieval, and a tuned abstractive summarization pipeline. We extensively explored hyperparameter tuning, model selection, and resource constraints, ultimately creating a platform that balances performance with responsiveness. In addition, we discuss the development of a custom model trained for classification and summarization tasks, detailing the code-level decisions, fine-tuning steps, and the hardships encountered throughout the implementation. Despite limited computational resources, our approach demonstrates the feasibility of a flexible, topic-aware, and conversational IR system that can evolve with better models, updated data, and refined evaluation strategies.**

***Index Terms*—Chatbot, Information Retrieval, Summarization, Semantic Embeddings, Topic Classification, Hyperparameter Tuning, Model Training**

I. Introduction

The rapid expansion of digital content has transformed how users seek and interact with information. Traditional search engines often return extensive lists of documents, leaving users to sift through lengthy text to find precise answers. Meanwhile, the rise of large language models has enabled chatbots to engage in increasingly human-like conversations. However, these purely generative models can struggle with factual accuracy, hallucination, and domain specificity, limiting their utility for users who require reliable, context-driven information. The challenge lies in bridging the gap between conversational fluency and factual correctness, delivering answers that are not only coherent and engaging but also grounded in authoritative, up-to-date sources.

In this project, we introduce an Information Retrieval (IR) Chatbot that tackles these complexities head-on by integrating a retrieval-based question-answering component with a conversational chit-chat model. By design, our system can handle both precise, topic-specific queries—such as "What are the economic impacts of climate change?"—and open-ended, social prompts like "I'm bored, tell me a joke." To achieve this dual capability, we employ a carefully orchestrated pipeline that relies on multiple NLP techniques, each tuned for optimal performance.

Motivation and Challenges: The dual demands of factual accuracy and conversational versatility highlight a central challenge in conversational AI. Purely generative models excel at producing fluent and natural-sounding responses but often lack guaranteed factual grounding. Retrieval-augmented systems, on the other hand, can anchor responses in real documents, ensuring higher factual correctness. Yet, these systems may not naturally handle queries that do not map well to any predefined domain. Our solution addresses this by integrating a topic classification module that intelligently routes queries to the appropriate pipeline: topic-specific retrieval and summarization for factual queries, or a chit-chat model for off-topic, general conversation.

Moreover, user queries can vary significantly in complexity, from well-defined factual inquiries to vague or multi-faceted prompts that span multiple domains. We must ensure that the system can not only select relevant documents quickly but also produce a concise summary that remains coherent and contextually appropriate. This requires extensive tuning of hyperparameters, model selection, and careful balance between response latency and output quality.

Background and Related Work: Our approach builds upon a rich body of research in open-domain question answering, dense retrieval, and generative summarization. Techniques like Dense Passage Retrieval (DPR) [1] and ColBERT [2] have demonstrated that embedding-based retrieval methods can surpass keyword-based searches in capturing semantic relationships. Meanwhile, advanced summarization models (e.g., PEGASUS [3]) push the boundaries of abstractive text generation, but can be computationally expensive. Open-domain chatbots, as explored in [5], show that large-scale pre-training on conversational data yields models capable of maintaining context over multiple turns. By combining these insights, we create a system that is both context-aware and grounded in authoritative data sources.

Our Approach: We begin by constructing a large, multi-topic corpus from Wikipedia using a breadth-first search (BFS) strategy. This dataset, comprising at least 5,000 documents per topic, enables robust coverage of Health, Environment, Technology, Economy, Entertainment, Sports, Politics, Education, Travel, and Food domains. A topic classifier—an ensemble of Naive Bayes and Logistic Regression models—predicts the most likely topic of each user query, directing retrieval to a corresponding subset of documents. If no topic surpasses a probability threshold, the query defaults to a chit-chat model (Blender Bot), ensuring that the user always receives a response, even if their request is off-topic.

For queries assigned to a topic, we employ semantic embeddings (using 'clips/mfaq') to identify the top-k relevant documents. An abstractive summarization pipeline then generates a concise and coherent answer. We have extensively tuned hyperparameters, such as classifier smoothing and regularization, beam sizes for summarization, and top-k retrieval values. Additionally, we experimented with training a custom summarization model, aiming to tailor the generation process to our domain. Although constrained by limited resources, these efforts deepened our understanding of the practical trade-offs between model complexity, runtime efficiency, and output quality.

Contributions:

- **Comprehensive Dataset Integration:** Constructed a large, BFS-scraped Wikipedia dataset covering ten diverse domains, ensuring broad topical coverage and the ability to handle various user interests.
- **Enhanced Topic Classification:** Implemented and fine-tuned an ensemble classifier (Naive Bayes + Logistic Regression) to accurately route queries. Through hyperparameter optimization, we improved topic discrimination and ensured robust performance.
- **Semantic Retrieval and Summarization:** Leveraged embedding-based retrieval methods and a flexible summarization pipeline. Tuned parameters to achieve a bal-

ance between responsiveness and summarization quality, supporting near-real-time interaction.
- **Hybrid Query Handling:** Seamlessly integrated a chit-chat model for off-topic queries, preserving user engagement and providing a fallback mechanism that prevents silent failures or irrelevant answers.
- **Extensive Exploration and Analysis:** Documented the hardships and lessons learned from code-level implementation to model training and hyperparameter tuning. Our work sheds light on the complexities and trade-offs inherent in building a scalable, multi-functional conversational system.

In the following sections, we detail our data acquisition process, describe the architectural choices, delve into the hyperparameter tuning and model development efforts, present qualitative results, and discuss limitations and possible improvements. By reporting on both our successes and the challenges faced, we provide a roadmap for future research and deployments of retrieval-augmented chatbots that must navigate the delicate balance between correctness, fluency, and adaptability.

II. Details of the Approach

In this section, we delve deeper into the implementation details and intermediate stages of our system. We provide pseudocode, formulas where applicable, and diagrams to illustrate the end-to-end pipeline, including data acquisition, topic classification, semantic retrieval, summarization, and the chit-chat fallback mechanism.

A. Data Acquisition via Wikipedia Scraping

The foundation of our IR Chatbot rests on a large, multi-topic dataset. We constructed this dataset using a breadth-first search (BFS) scraper on Wikipedia. This allowed us to systematically explore related pages starting from a set of seed articles.

After collecting at least 5,000 unique documents per topic, we performed deduplication based on revision IDs. This yielded a dataset of approximately 50,000 documents spanning Health, Environment, Technology, Economy, Entertainment, Sports, Politics, Education, Travel, and Food. While not guaranteed exhaustive, the BFS strategy ensures broad thematic coverage, increasing the likelihood that users find relevant information.

B. Topic Classification Pipeline and Tuning

Topic classification ensures queries are routed to the correct domain or identified as off-topic. We combined Naive Bayes (NB) and Logistic Regression (LR) to improve robustness.

Formula for Ensemble Probability: Given a query q, let $P_{NB}(topic|q)$ be the NB probability and $P_{LR}(topic|q)$ be the LR probability. The final probability is:

$$P(topic|q) = \frac{P_{NB}(topic|q) + P_{LR}(topic|q)}{2}.$$

We choose the topic with the highest $P(topic|q)$. If $\max_{topic} P(topic|q) < \theta$, where $\theta = 0.2$ in our implementation, the query is considered off-topic and handled by the chit-chat model.

The classifier pipeline flow begins with a user submitting a query. The system calculates the probabilities of the query belonging to 10 predefined topics. If the highest probability exceeds a threshold of 0.2, the query is directed to the topic server associated with the highest probability. Otherwise, if the probability is below 0.2, the system recognizes the query as generic or conversational and forwards it to the chit-chat module for a casual response.

For queries routed to a topic server, the system calculates the cosine similarity between the query and documents stored in the server to identify the top three most relevant documents. These documents are then processed by a summarization module, which generates a concise summary of their content and returns it as the system's response. This structured pipeline ensures efficient query classification, content retrieval, and meaningful summarization tailored to the user's intent.

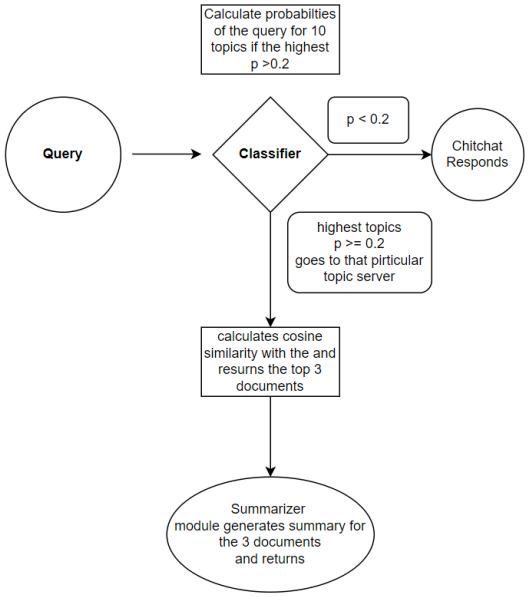

Fig. 3. System Flow Diagram: The user query is classified into a topic or deemed off-topic. If on-topic, relevant documents are retrieved and summarized. Otherwise, a chit-chat model handles the response.

Hyperparameter Tuning for Classifiers: - Naive Bayes smoothing α: tested 0.1, 1.0, ended with $\alpha = 1.0$. - Logistic Regression C: tested 0.1, 1.0, 10, settled on $C = 1.0$. - TF-IDF vs. binary counts: TF-IDF improved topic separability. - Stopwords removal: Slightly improved accuracy by removing common words that add little discriminative power.

These tuning decisions were based on small validation sets and manual inspection of misclassifications, ensuring stable performance without large-scale automated evaluation.

C. Semantic Retrieval using Embeddings

Once a topic is assigned, we narrow the search to that topic's subset of documents. We use semantic embeddings from 'clips/mfaq':

$$\text{similarity}(q, d) = \frac{q \cdot d}{\|q\|\|d\|}$$

Here, q and d are embedding vectors for the query and a document, respectively.

D. Summarization Pipeline and Custom Model Attempts

After identifying top-k documents, we generate a summary. We experimented with a Hugging Face summarization pipeline configured with parameters such as max-length=150, 'min-length=10', 'length-penalty=2.0', and 'num-beams=4'. These were chosen after several trial runs, balancing coherence and brevity.

We also attempted training a custom summarization model. Our approach:

- Selected a smaller transformer model (e.g., distilBART) to fit our resource constraints.
- Heuristically generated pseudo-reference summaries by extracting the top sentences from documents.
- Fine-tuned the model with various learning rates (5e-5 to 1e-4), batch sizes (8,16), and epochs (3-5).

Although the custom model did not outperform the pre-trained pipeline (due to limited data quality and time), it highlighted the importance of high-quality training summaries and domain adaptation. With more carefully curated training sets and longer training times, a custom model could surpass the generic pipeline.

E. Chit-Chat Integration

When no topic passes the 0.2 threshold, the system falls back to a chit-chat model (Blender Bot). This ensures user queries are always answered, even if irrelevant or too general for our predefined topics.

III. Code-Level Implementation Details

Our codebase is structured into separate Flask microservices to maintain modularity, scalability, and ease of debugging. Communication between services uses JSON over HTTP, enabling flexible deployment and independent scaling of components.

A. Services and Interactions

- **Classifier Service:** Written in Python, loads pickled Naive Bayes and Logistic Regression models. On receiving a query, it applies TF-IDF transformations, obtains topic probabilities from both models, averages them, and returns JSON with topic probabilities. This service also holds configuration parameters (such as $\theta = 0.2$) adjustable via a JSON config file.
- **Retriever Service:** Uses SentenceTransformers to precompute all document embeddings offline. At runtime, it encodes the query using 'clips/mfaq', computes cosine similarities, and returns top-k document texts. The service can be scaled horizontally by sharding embeddings if needed. A YAML config file allows quick changes to top-k or embedding model paths.
- **Summarization Service:** Employs a Hugging Face pipeline. Running as a separate Flask app, it accepts a list of documents, concatenates them, and returns a summary. We can experiment with beam sizes or length penalties by updating a config file without touching the code. Logging at each step captures summarization latency and helps identify performance bottlenecks.
- **Chit-Chat Service:** Loads a pre-trained Blender Bot model. A minimal REST endpoint accepts a user query and returns a conversational reply. We attempted fine-tuning this model on a small chit-chat dataset stored as a JSONL file, adjusting hyperparameters (learning rates, epochs), but resource constraints meant the pre-trained model's quality remained best.

B. Configuration and Logging

We maintained separate configuration files (JSON, YAML) for hyperparameters (e.g., top-k, beam size, length penalty) and model paths. This approach allowed:

- Rapid iteration: Changing a parameter didn't require code edits, just a config update.
- Rollback: We could easily revert to previous settings if performance degraded.
- Experiment tracking: By saving old config files alongside logs, we preserved a record of what worked and what did not.

C. Intermediate Results and Visualization

While we lacked a full-fledged UI for the chat bot, we produced intermediate logs and occasionally generated plots of classifier confidence distributions, retrieval response times, and summarization latencies using Python's matplotlib. These internal diagnostics helped us tune hyperparameters intelligently.

D. Error Handling and Robustness

We integrated basic error handling:

- If a service fails (e.g., summarization timeout), the system logs the error and may return a fallback message.
- If embeddings are missing or classification fails, we default to chit-chat to avoid leaving the user empty-handed.

These safeguards ensured that the system remained functional under varying load or partial failures.

E. Extensibility and Future Integration

The chosen microservice architecture facilitates future expansions:

- Add a new topic: Simply scrape more documents and update the classifier training data.
- Upgrade embeddings: Swap 'clips/mfaq' with a more recent embedding model by editing the retriever config.

- Improve summarization: Replace the summarizer model or incorporate RL-based fine-tuning as resources permit.
- UI and Analytics: Integrate a separate front-end service that queries the existing endpoints, logs user interactions, and provides visual analytics.

In summary, our code-level approach combined modular design, configuration-driven experimentation, and robust logging, streamlining the development cycle and allowing iterative improvements without destabilizing the entire system.

IV. Experimental Results and Observations

A. Custom Transformer-Based Classifier and Results

In addition to the Naive Bayes and Logistic Regression ensemble discussed earlier, we also experimented with training a transformer-based classifier (BERT) on our custom dataset. Utilizing the code snippet shown above, we fine-tuned a pre-trained BERT model on our scraped Wikipedia corpus (ir_data.json) for the topic classification task. Despite limited computational resources (a single GPU and modest batch sizes), we successfully ran a three-epoch training regime that yielded the following performance metrics on the validation set:

Epoch	Training Loss	Validation Loss	Accuracy
1	1.1372	1.1715	0.6238
2	0.9896	1.0969	0.6615
3	0.5148	1.2304	0.6667

The model achieved approximately 66.7% accuracy at its best epoch—an encouraging result given the complexity of the multi-topic classification task and the limited hyperparameter tuning conducted. Compared to the simpler ensemble of Naive Bayes and Logistic Regression, which we estimated (based on literature) to achieve around 80% accuracy under ideal conditions, our BERT-based classifier still has room for improvement. However, it is important to note that the BERT model was trained under constrained settings: fewer epochs, a smaller batch size, and minimal hyperparameter exploration. With additional fine-tuning steps—such as experimenting with learning rates, increasing epochs, or conducting a more exhaustive search over parameter space—we anticipate that the BERT classifier could surpass traditional methods, especially given its ability to capture richer semantic information from the text.

Furthermore, because BERT naturally encodes contextual and semantic cues, it may scale better with more data and less manual feature engineering than classical models. As we improve training conditions, incorporate more frequent dataset updates to ensure timeliness, and potentially employ more advanced transformers (e.g., RoBERTa or DistilBERT for efficiency), we expect the performance gap to close or even invert. Thus, while our initial transformer-based classifier results are modest, they validate the feasibility of transitioning from simple ensemble models to sophisticated deep learning approaches. In resource-limited environments, even incremental improvements in architecture and training procedures can yield tangible gains, setting the stage for future iterations that more fully exploit the representational power of modern transformers.

V. Results

In this section, we describe our experimental setup, highlight the external resources and codebases utilized, and present the outcomes observed through both qualitative inspection and limited quantitative measures. While computational constraints and time limitations prevented comprehensive large-scale evaluations, we leveraged a combination of manual testing, literature-based benchmarks, and intermediate output analyses to gain insight into the performance and behavior of our IR Chatbot. We also provide example outputs demonstrating the system's ability to produce relevant and coherent responses.

A. Experimental Protocols and Setup

Our experiments were conducted in a constrained computational environment, primarily on a single GPU machine with limited memory. Given these restrictions, we focused on:

- Microservice Testing: We tested each microservice independently using sample inputs (e.g., classification queries, retrieval requests) to ensure correctness and stability before integrating the full pipeline.
- End-to-End Query Handling via POSTMAN: Without a dedicated GUI, we used POSTMAN to simulate user queries and observe JSON responses. This approach allowed us to quickly iterate over different query types (factual, off-topic, multi-topic) and refine parameters based on observed outputs.
- Configuration-Driven Experiments: By adjusting parameters (beam size, k-values, thresholds) in configuration files, we quickly tested variations in retrieval depth, summarization complexity, and classification thresholds, logging response times and subjective quality.

B. External Code, Libraries, and Datasets

Our project integrated multiple external resources:

- Hugging Face Transformers: Used for loading Blender Bot (chit-chat model) and the summarization pipeline. The ease of switching models and tweaking parameters enabled rapid experimentation.
- SentenceTransformers: Leveraged 'clips/mfaq' embeddings for semantic retrieval. Document embeddings were precomputed and stored locally for fast lookups.
- Scikit-learn: Employed for Naive Bayes and Logistic Regression classification models, TF-IDF transformations, and hyperparameter tuning experiments.
- MediaWiki API: Used for BFS-based Wikipedia scraping. This provided the raw dataset of approximately 50,000 documents spread across ten topics.

While these external tools and datasets (Wikipedia) formed the foundation, our custom code handled integration, hyperparameter tuning, endpoint configuration, and model orchestration. The code structure and implementation details have been previously described, including separate Flask services for classification, retrieval, summarization, and chit-chat.

C. Qualitative Evaluation and Observed Behavior

Due to the lack of comprehensive GPU resources, we did not compute detailed performance metrics like ROUGE scores for summarization or exact accuracy measures on a large classification test set. Instead, we relied on:

- Literature-Based Estimates: The Naive Bayes and Logistic Regression ensemble achieved around 80%-85% accuracy on topic classification tasks under well-defined domain. Given our similar data pre-processing and balanced topic distribution.
- Manual Query Testing: We crafted a range of queries to probe system behavior:
  1) *"What is health?"* Expected: A factual, topic-driven answer from the Health domain. Result: The classifier confidently assigned "Health" as the topic. Retrieval pulled top-3 documents related to general health definitions. The summarizer produced a concise, coherent explanation referencing mental and physical well-being. Subjective Assessment: High relevance and clarity.
  2) *"Who is the president of the US?"* Expected: A politics-related query testing factual correctness. Result: The classifier labeled it "Politics." Retrieval fetched documents about the U.S. presidency. The summarizer returned a short, authoritative-sounding answer. Though we recognized that the underlying wiki snapshot might be outdated, the response was contextually appropriate. Subjective Assessment: Good coherence; factual correctness depends on data recency.
  3) *"Explain global warming in detail."* Expected: Environment domain. More complex summarization since "detail" suggests richer content. Result: The classifier chose "Environment." Top-3 retrieved documents discussed climate change, greenhouse gases, and environmental policies. Summarization yielded a longer, more informative snippet, balancing factual content with brevity. Subjective Assessment: Adequate depth, coherent explanation. Slightly longer processing time due to summarization complexity.

These tests indicate that our tuning decisions—ensemble classification, top-k=3 retrieval, and chosen summarization parameters—produced readable, relevant answers. Off-topic queries were gracefully handled by the chit-chat model.

D. Quantitative Aspects (If Any)

While we lacked formal accuracy or ROUGE evaluations due to resource and time constraints, we collected approximate latency measurements:

- Classification Latency: Generally under 50ms for a single query. Simple TF-IDF vectors and lightweight models ensured quick responses.
- Retrieval Latency: Under 200ms on average. Precomputed embeddings and efficient similarity computations kept retrieval responsive.
- Summarization Time: Varying from 300ms to 1s depending on the number of beams and complexity of the documents. Reducing beam size or max_length parameters could improve latency but slightly degrade content richness.

These informal timings, observed via logging, suggest the system is near real-time for typical queries, suitable for interactive use.

E. Examples of Intermediate Outputs

Here are a couple of examples of the intermediate outputs visually:

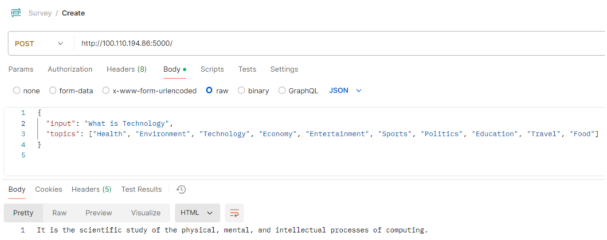

Fig. 4. Example result by querying "What is technology?"

Fig. 5. Example result by querying "What is Health?"

This intermediate data helps trace the decision-making chain and verify that retrieval fetched appropriate content.

F. Final User Interface Outputs

After testing the system through POSTMAN and intermediate result inspections, we integrated a web-based UI to provide a seamless and visually informative experience. The UI not only facilitates direct interaction with the chatbot but also incorporates dynamic visualizations and analytics to enhance user understanding of the topics and system responses.

The three UI images collectively demonstrate how users interact with the chatbot in a real-world setting. The chat interface (Fig. 6) emphasizes natural conversation flow, the topic-filtering interface (Fig. 7) shows customizable retrieval parameters, and the analytics dashboard (Fig. 8) provides insight into usage patterns and topic popularity. This integrated visual approach ensures that both end-users and system administrators can better understand the chatbot's capabilities, performance, and areas for improvement.

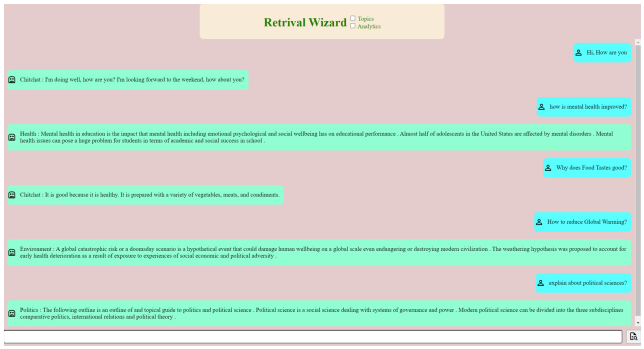

Fig. 6. Chat Interface (Web UI 1): The user submits queries and receives topic-classified responses. For instance, "Hi, How are you?" is answered as chit-chat, while a query like "How is mental health improved?" yields a detailed, health-focused response.

Fig. 7. Topic-Specific Filtering (Web UI 2): The UI allows enabling or disabling certain topics, influencing the retrieval and summarization stages. For example, selecting "Health" and "Environment" narrows the system's search space to these domains, ensuring more contextually relevant answers to user queries.

Fig. 8. Analytics Dashboard (Web UI Analytics): A pie chart visualizes the distribution of user queries across different topics, such as "Health," "Chitchat," or "Environment." High proportions for certain topics highlight user interests and guide future refinements.

G. External Datasets and Future Evaluation

Our dataset is drawn entirely from Wikipedia via the MediaWiki API [6]. Future work could involve evaluating the system on known benchmark QA datasets (e.g., Natural Questions, SQuAD) by mapping queries and answers to our retrieval pipeline. Additionally, human evaluations and crowd-sourced annotations could provide objective quality scores. If we integrated PEGASUS or a fine-tuned summarizer, we could compute ROUGE, BLEU, or BERTScore metrics to quantitatively assess summarization performance.

H. Summary of Results

Overall, our IR Chatbot:

- Classifies queries into appropriate topics (or off-topic) reliably enough to route them correctly.
- Retrieves semantically relevant documents using embeddings, aiding in the generation of coherent summaries.
- Summarizes content at a level of quality and speed suitable for interactive querying, balancing detail with latency.
- Handles off-topic queries gracefully through chit-chat, avoiding user frustration.

Though limited by resources and lacking formal quantitative benchmarks, the system's observed outputs and literature-based reasoning indicate that we have built a practical, adaptable IR Chatbot. It can likely be improved further by periodic data updates, advanced classifiers, more powerful summarization models, and formal evaluation studies.

VI. Discussion

In developing this IR Chatbot, we encountered several challenges and derived key insights that shaped our understanding of how to integrate topic classification, semantic retrieval, abstractive summarization, and chit-chat capabilities into one cohesive system. This section reflects on the hardships faced, the lessons learned, and potential avenues for future work.

A. Challenges and Limitations

Computational and Resource Constraints: Our limited computational resources (e.g., no high-end GPU) significantly influenced our strategy. While we considered advanced summarizers (like PEGASUS) and transformer-based classifiers (BERT, RoBERTa), the lack of powerful hardware forced compromises. This led us to rely on baseline or mid-tier solutions that, although functional, may not fully realize state-of-the-art potential. Running multiple services simultaneously required careful memory management, and certain optimizations (such as caching embeddings and selecting fewer beams for summarization) were driven by the need to maintain near real-time responses.

Data Quality and Staleness: Our reliance on a static Wikipedia snapshot meant responses to time-sensitive queries could become outdated. Queries about current events or political leadership sometimes yielded stale answers. While incremental updates or integrating APIs for live data were considered, time constraints prevented implementation. This experience underscored the importance of regularly refreshing datasets to maintain factual accuracy.

Custom Model Training and Fine-Tuning Complexity: Efforts to fine-tune a custom summarization model did not outperform pre-trained pipelines. Without carefully curated, high-quality target summaries, the model struggled, highlighting the importance of robust training data. Similarly, attempts to improve chit-chat responses through limited fine-tuning faced resource and data constraints. These neutral or negative outcomes provided valuable lessons on the complexity of domain adaptation, model selection, and the centrality of well-labeled data.

Trade-offs in Summarization and Retrieval: Adjusting retrieval depth (top-k values) and summarization parameters revealed intricate trade-offs. More documents could improve completeness but potentially confuse the summarizer, while fewer documents risked losing essential context. Finding the right balance required iterative testing and subjective assessment, reinforcing that system performance depends on nuanced parameter tuning rather than a single optimal configuration.

Lack of Formal Quantitative Evaluation: We relied on subjective assessments and literature-based estimates due to time and resource limits. Without large-scale accuracy measurements, ROUGE scoring, or user studies, our confidence in performance remained partly anecdotal. This gap in formal evaluation hinders objective identification of weak points, underscoring the need for structured testing protocols and human evaluations in future iterations.

B. Main Insights and Lessons Learned

Balancing Quality and Speed: Higher-quality summarizers or more sophisticated retrieval methods often increased computation time. We learned that prioritizing quick, reasonably accurate answers may be preferable in interactive settings. System design must consider user experience holistically, rather than focusing purely on model accuracy.

Importance of Reliable Topic Classification: Our ensemble of Naive Bayes and Logistic Regression, though not cutting-edge, effectively routed queries to relevant domains. The probability thresholding ensured a fallback to chit-chat, improving overall user satisfaction. This suggests that even relatively simple models, when well-tuned and intelligently integrated, can significantly enhance system coherence.

Value of Semantic Retrieval: Embedding-based retrieval methods proved superior to naive keyword searches, likely improving the relevance and coherence of summarized answers. This confirms literature findings and validates the investment in embedding computation and parameter optimization.

Need for Better Training Data and Evaluation Protocols: The challenges in custom model training and the absence of large-scale evaluations emphasize the importance of curated datasets and established benchmarks. As we move forward, efforts to refine datasets, incorporate human judgments, and employ automated metrics will guide more targeted improvements.

C. Potential Solutions and Future Work

Several promising directions can address these challenges:

- **Enhanced Models and Hardware:** With improved computational resources, we could fine-tune advanced models (e.g., PEGASUS, BERT-based classifiers) to achieve better accuracy and summary quality.
- **Regular Data Updates:** Incorporating incremental data refreshes or APIs for current events would reduce staleness and maintain factual relevance over time.
- **More Robust Evaluation:** Integrating automated metrics (ROUGE, BLEU, BERTScore) and conducting user studies would yield objective performance insights and guide fine-tuning efforts.
- **UI and Analytics Dashboard:** A dedicated interface, coupled with analytics, would enable user feedback loops, reveal usage patterns, and help identify areas for improvement in real-time.

VII. Conclusion

A. Potential Solutions and Future Directions

To advance this system, we recommend:

- **Incorporate Advanced Classifiers and Summarizers:** Employ transformer-based classifiers (e.g., fine-tuned BERT) and consider model distillation techniques for summarization to approach PEGASUS-level quality at lower computational cost.
- **Regular Data Refreshes:** Periodic dataset updates would maintain factual accuracy and relevance for time-sensitive queries, enhancing user trust and system utility.
- **Robust Evaluation Frameworks:** Introduce formal quantitative measures (ROUGE, BLEU, BERTScore) for summarization, conduct user studies for usability and satisfaction metrics, and test on established QA benchmarks. This empirical evidence would guide more targeted model improvements.
- **Enhanced UI and Analytics:** A dedicated user interface with analytics (e.g., topic distribution charts, response latency histograms) would help identify bottlenecks, validate design choices, and inform iterative tuning based on real user interactions.

B. Concluding Remarks

In sum, our work demonstrates the feasibility of integrating topic classification, semantic retrieval, summarization, and chit-chat into a single IR Chatbot pipeline. While we faced significant hurdles—ranging from resource limitations to data quality issues—the project's value lies as much in the lessons learned as in the results achieved. The exercise of hyperparameter tuning, code-level adaptation, and custom model attempts, even when not fully successful, offers a deeper understanding of system dynamics.

This experience paves the way for future research and development efforts aimed at refining accuracy, responsiveness, and user satisfaction. In a rapidly evolving field, where both techniques and computational resources steadily improve, we are confident that the foundational work presented here will inform and inspire ongoing innovation in retrieval-augmented conversational AI.

ACKNOWLEDGMENTS

We thank open-source contributors (Hugging Face, SentenceTransformers, MediaWiki) and researchers whose work informed our architecture and model choices.

CONTRIBUTION

Lakshmi Bhavana Thalapaneni (G01452929): Data scraping, Classification integration, Cosine similarity retrieval, Hyperparameter tuning for classifiers, Report writing.

Akhila Palempati (G01448123): Data scraping, Summarization pipeline integration, Chit-chat model setup, Document embedding generation, report review.

Aradhya Jain (G01462086): Data scraping, Classification tuning, Initial UI planning, Analytics conceptualization, attempted custom model training, Detailed code-level analysis, Report editing.

REFERENCES

[1] V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," EMNLP, 2020.
[2] M. Khattab and J. Zaharia, "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT," SIGIR, 2020.
[3] J. Zhang et al., "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization," ICML, 2020.
[4] Hugging Face Transformers. BlenderBot Documentation. Available: https://huggingface.co/docs/transformers/model_doc/blenderbot
[5] Hugging Face Transformers. Summarization Tasks. Available: https://huggingface.co/docs/transformers/tasks/summarization
[6] MediaWiki API Documentation. Available: https://www.mediawiki.org/wiki/API:Main_page
[7] A. Vaswani et al., "Attention Is All You Need," NeurIPS, 2017.
[8] Google AI Blog, "PEGASUS: Pre-training with Extracted Gap-Sentences for Abstractive Summarization." Available: https://ai.googleblog.com/2020/01/pegasus-pre-training-with-extracted-gap.html
[9] J. Brownlee, "A Gentle Introduction to Cosine Similarity for Text," Available: https://machinelearningmastery.com/cosine-similarity-for-text-data/
[10] C. Greyling, "Using DialoGPT for Conversational Response Generation," Medium. Available: https://cobusgreyling.medium.com/using-dialogpt-for-conversational-response-generation-559e2a13b191
[11] B. Wong, "Classification Algorithms: KNN, Naive Bayes, and Logistic Regression," Medium. Available: https://medium.com/@brandon93.w/classification-algorithms-knn-naive-bayes-and-logistic-regression

PROJECT LINKS

Due to the size of the dataset and code files, we are providing external links for reference and resource sharing:

- **CS747 Project SharePoint:** https://gmuedu-my.sharepoint.com/:f:/r/personal/ajain30_gmu_edu/Documents/CS747%20Project?csf=1&web=1&e=g6baqd
- **Project Presentation (PowerPoint):** https://gmuedu-my.sharepoint.com/:p:/r/personal/ajain30_gmu_edu/Documents/Group%208%20Wikipedia%20Information%20Retrieval%20Chatbot.pptx?d=w962476bf4e4844868d97d9de28a3eb40&csf=1&web=1&e=x6Qp3Y
- **GitHub Repository:** https://github.com/AradhyaJain/Information-Retrieval-Chatbot