**Titan Company NLP Project Report (FY 2023–24)**

**Abstract**

This project builds a lightweight Retrieval-Augmented Generation (RAG) assistant to answer investor- and company-related questions using the Titan Company Annual Report FY 2023–24 and a Titan FAQ document. The system combines classical information retrieval (TF–IDF with cosine similarity) for fast, transparent context selection and a generative model (via OpenRouter) for fluent, grounded answers. The app is delivered as a Streamlit interface with configurable parameters and explainability through displayed source snippets.

**Objectives**

* Provide accurate, grounded answers strictly from Titan’s authoritative documents.
* Offer explainability by surfacing the exact passages used to answer.
* Keep the solution lightweight and easily deployable on modest hardware.
* Support multiple corpora (Annual Report + FAQ PDF) and flexible retrieval settings.

**Documents and Data Sources**

* Annual Report: Titan Annual Report 2023-24 - 21MB 3.pdf (preprocessed into titan\_chatbot\_data.json with pages, vocabulary, TF–IDF matrix)
* FAQs: Titan\_Company\_FAQ.pdf (parsed at runtime, vectorized against the same vocabulary)

**Methodology**

1. Preprocessing (Annual Report)
   * Segment PDF into pages and clean text.
   * Build a vocabulary and TF–IDF matrix offline.
   * Persist artifacts into titan\_chatbot\_data.json for quick loading.
2. Runtime Retrieval (Annual Report + FAQ)
   * Build a CountVectorizer with the fixed vocabulary to encode queries.
   * Encode the FAQ pages into a TF–IDF matrix aligned with the same vocabulary at app startup.
   * For a user query, compute cosine similarity against both matrices and select top‑k passages.
3. Answer Generation
   * Construct a prompt with the top passages (source and page metadata included).
   * Call OpenRouter (configurable model; default openai/gpt-3.5-turbo).
   * Enforce groundedness: the system prompt instructs the model to only use provided context and avoid fabrication.

**System Architecture**

* UI Layer: Streamlit app (titan\_faq\_app.py) with search input, Ask button, Quick FAQs, and a context viewer.
* Retrieval Layer: TF–IDF + cosine similarity over Annual Report and FAQ corpora.
* Generation Layer: OpenRouter API integrates a hosted LLM for answer synthesis.
* Caching: Streamlit resource caching for loaded artifacts and FAQ matrix.

**Implementation Overview**

* File: titan\_faq\_app.py
  + Loads titan\_chatbot\_data.json (pages, vocabulary, TF–IDF matrix for Annual Report).
  + Builds a fixed‑vocabulary CountVectorizer for query encoding.
  + Parses Titan\_Company\_FAQ.pdf using pypdf (fallback to PyPDF2) and creates a TF–IDF matrix for the FAQ corpus.
  + Retrieves top‑k snippets from both sources, merges and sorts by score, and limits to k.
  + Displays the answer and the exact supporting passages with source labels (Annual Report or FAQ PDF).
  + Includes a horizontally centered Quick FAQs section to prefill common queries.
  + Adds a simple centered footer.

**Key Dependencies**

* Python 3.10+
* streamlit, numpy, scikit-learn, requests, pypdf or PyPDF2

**Running the App**

py -m pip install -r titan\_requirements.txt *# if available*

py -m pip install streamlit numpy scikit-learn requests pypdf PyPDF2

streamlit run titan\_faq\_app.py

**Configuration**

* API: OpenRouter key can be set in code or overridden via the sidebar.
* Retrieval: k (number of snippets) and MAX\_SNIPPET\_CHARS are adjustable from the sidebar.

**Evaluation**

We evaluate along three dimensions:

* Retrieval Quality: Are the top snippets relevant and sufficient? Manual spot checks across representative investor and operations queries.
* Answer Groundedness: Does the model refrain from inventing content not present in the provided snippets?
* UX: Latency, clarity of “Context used,” and usability of Quick FAQs.

Indicative Test Queries

* “Summarize FY 2023–24 performance highlights.”
* “Has the company declared dividends this year?”
* “What are Titan’s key business segments?”
* “Who should I contact for investor relations?”
* “What is the shareholding pattern?”

Success Criteria

* The answer cites correctly aligned passages; users can verify facts in the displayed context.
* Minimal hallucinations; the assistant explicitly says when the answer is not in context.

**Results (Illustrative)**

* The merged retrieval from Annual Report and FAQ improved coverage for investor‑centric questions.
* Displaying source metadata (Report vs FAQ) increases trust and traceability.
* Quick FAQs reduced time‑to‑answer for common queries.

**Conclusion**

This project delivers a practical, explainable RAG assistant for Titan Company by combining fast TF–IDF retrieval with an LLM to generate grounded answers. Merging the Annual Report with the FAQ corpus improves coverage for investor‑focused questions while keeping the system lightweight and easy to deploy. The app’s “Context used” view increases trust through transparent citations, and the Quick FAQs streamline common queries. With the proposed enhancements (dense retrieval, reranking, and richer citations), the system can further improve recall and user confidence while maintaining simplicity.

**Ethical and Compliance Notes**

* Answers are restricted to supplied corporate documents; the assistant does not provide financial advice.
* Users are informed when requested information is not present in the provided context.

**References**

* Titan Company Annual Report 2023–24
* Titan Company FAQ document 2023–24
* OpenRouter API documentation (https://openrouter.ai/docs)
* Scikit‑learn documentation (https://scikit-learn.org/)
* Streamlit documentation (https://docs.streamlit.io/)

**Appendix A: Environment and Variables**

*# OpenRouter*

OPENROUTER\_API\_KEY=<your\_key>

**Appendix B: High‑Level Pseudocode**

load pages, vocabulary, tfidf\_matrix from titan\_chatbot\_data.json

vectorizer = CountVectorizer(vocabulary)

faq\_pages = read\_pdf(FAQ\_PDF\_PATH)

faq\_matrix = TfidfTransformer().fit\_transform(vectorizer.transform(text\_of(faq\_pages)))

def retrieve(query):

q = normalize(vectorizer.transform([query]))

s1 = cosine\_similarity(q, matrix).argsort(desc)[:k]

s2 = cosine\_similarity(q, faq\_matrix).argsort(desc)[:k]

return top\_k\_by\_score(merge(s1\_pages, s2\_pages), k)

snippets = retrieve(user\_query)

prompt = build\_prompt(snippets, user\_query)

answer = openrouter(prompt)

display(answer, snippets)