Assignment Report: Comparative Financial QA System (RAG vs Fine-Tuning)

1. Data Collection & Preprocessing

- **Data Sources**: Downloaded two real-world annual reports (Infosys AR 2024 and 2025).
- **Extraction**:
  - Implemented `extract_pdf_text()` with **PyMuPDF** for robust text extraction.
  - Added **clean_lines()** to strip headers, footers, and junk text.
  - Stored outputs in `.txt` and `.pages.json` for reuse.
- **Segmentation**:
  - Defined improved `SECTION_PATTERNS` (Income, Balance Sheet, Cash Flow, MD&A, Notes).
  - Tagged each page with its section to improve downstream retrieval.
- **Chunking**:
  - Created overlapping word chunks (`100` and `400` tokens) with `make_chunks()` for retrieval experiments.
  - Normalized mapping of words → pages for back-referencing.
- **Q/A Dataset Construction**:
  - Extracted **numeric metrics** (Revenue, Net Profit, EBITDA, EPS, Assets, Liabilities, Cash Flow, Margins, Headcount, Equity, R&D expense, CapEx, OpEx).
  - Extracted **textual facts** (CEO, CFO, Auditors, HQ, Segments).
  - Added **YoY comparison Q/As** (e.g., revenue 2024 vs 2023).
  - Balanced to ~50 Q/As (numeric, textual, comparison, and control/irrelevant queries).

Output: `qa_pairs.jsonl` and `qa_pairs.csv`

---

2. RAG System Implementation

2.1 Embedding & Indexing

- Used **all-MiniLM-L6-v2** for dense embeddings.
- Built **FAISS index** (cosine similarity, normalized vectors).
- Built **BM25** sparse index with custom tokenizer and stopword filter.

2.2 Hybrid Retrieval

- Combined dense + sparse results using **Reciprocal Rank Fusion**.
- Added **Cross-Encoder (MiniLM MS MARCO)** for re-ranking top-N passages.

2.3 Generation

- Used **Flan-T5-base** as generator.
- Built **prompt constructor** with strict token budget and guardrails:
  - Concatenate only as many passages as fit ≤512 tokens.
  - Instructions: *"Use ONLY context, don't hallucinate, report numbers exactly as written."*

2.4 Guardrails

- **Input guardrail**: Block irrelevant queries (capital of France).
- **Output guardrail**: Factuality check:
  - Extracted numbers + units from answer.
  - Compared with numbers in context (strict + lenient).
  - Flagged hallucinations → downgraded confidence / returned "Not in scope."

2.5 Improvements Made

- Query expansion (add FY tokens, revenue/profit hints).
- Minimal extractor fallback: regex-based answer extraction when generator is conservative.
- Dynamic `k_final`: wider context for numeric queries.

Output: `rag_answer()` function with hybrid retrieval, guarded generation, and factuality check.

---

3. Fine-Tuned Model (FT) System

3.1 Dataset

- Used the **same Q/A dataset** (~50 examples).
- Converted to `prompt` + `target` for supervised fine-tuning.

3.2 Model Selection

- **Flan-T5-base** chosen as baseline (balance of accuracy and efficiency).

3.3 Baseline Benchmarking

- Evaluated base model on 10 test Q/As.
- Metrics: Exact Match (EM), token-level F1, and latency.

3.4 Fine-Tuning

- Used **Hugging Face Trainer**:
  - Optimizer: AdamW

- Batch size: 8
    - Epochs: 10
    - Learning rate: 5e-5
    - Early stopping added (`load_best_model_at_end=True`).

3.5 Advanced FT Technique

- Simulated **Mixture-of-Experts** via parameter-efficient finetuning:
    - Modular adapters per metric domain.
    - Router selects expert during inference.
    - Compared **single FT** vs **MoE-LoRA** on EM, F1, latency.

3.6 Guardrails

- Added same **input/output filters** as RAG for robustness.

Output: Fine-tuned model stored in `fine_tuned_model/`.

---

4. Testing & Evaluation

- **Queries used**:
    - Relevant high-confidence (e.g., revenue 2024).
    - Relevant low-confidence (ambiguous queries).
    - Irrelevant (capital of France).
- **Evaluation table** included:
    - Model (RAG/FT)
    - Answer
    - Confidence
    - Response time
    - Correctness

4.1 Findings

- **RAG Strengths**:
    - More grounded → avoids hallucination.
    - Robust to irrelevant queries.
    - Slower (retrieval + re-ranking overhead).
- **FT Strengths**:
    - Faster inference.
    - More fluent answers.
    - Can hallucinate if query out of training distribution.
- **MoE-LoRA** improved:
    - Lower latency.
    - Slight boost in F1 on numeric queries.

---

5. Deliverables

- Code: Python notebook with RAG + FT pipelines.
- Data: `qa_pairs.jsonl` + `qa_pairs.csv`.
- Report (this document).
- Balanced Q/A dataset (~50 items).
- UI (Streamlit/Gradio) with switch between RAG and FT.
- Screenshots of sample queries and outputs.

Final Summary

We successfully:

- Built a **Retrieval-Augmented Generation (RAG)** pipeline with dense+BM25 hybrid retrieval, re-ranking, guarded generation, and factuality checks.
- Built a **Fine-Tuned (FT)** chatbot with baseline benchmarking, supervised fine-tuning, and an MoE-inspired advanced method.
- Created a **balanced evaluation dataset** from real financial reports.
- Compared both methods in terms of **accuracy, speed, robustness, and trade-offs**.