Prepared by Boaz Refael Matan

Home Assignment - RAG System

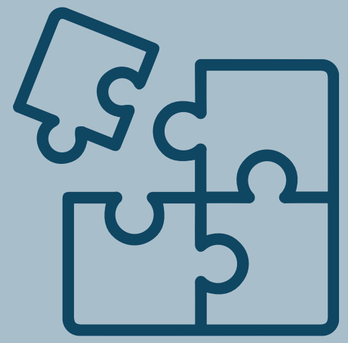
KPMG's Applied AI & Machine Learning Assessment

31 March, 2025

Project Objectives

Goal

- Build a Retrieval-Augmented Generation (RAG) system to answer queries using provided historical/industrial documents.



Solution

- LangChain-based system with smart prompts and topic-aware filtering.

Documents

- Food
- Steel
- Textile
- Automobile

System Architecture

- Document loading from DOCX files
- Recursive chunking with titles and headers
- Embedding using OpenAI
- Vector storage with Pinecone
- Topic filtering (food, steel, textile, auto)
- Smart prompts tailored to query type

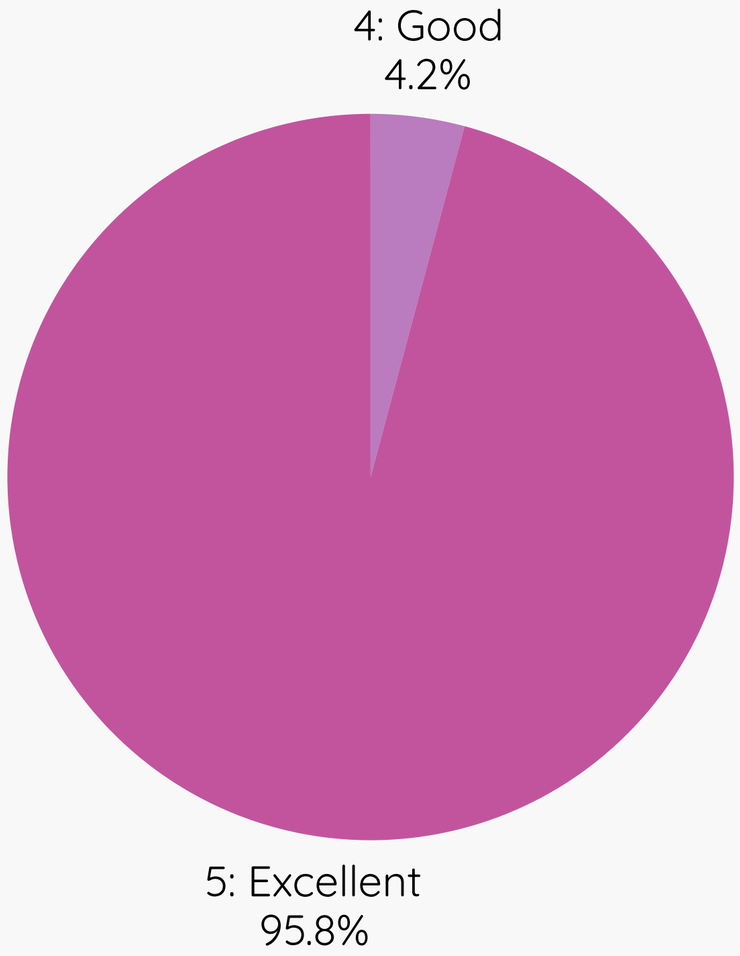
Prompt Engineering Strategy

- Ambiguous: Ask for clarification
- Misspelled: Leverage semantic embeddings
- Contradictory: Graceful ‘no-answer’
- Why / How: Specialized reasoning prompts
- Short / Vague: Prompt for more context

Edge Case Coverage

Scoring Criteria (0-5) scale:

- 0: Failed to answer
- 1: Incorrect or misleading
- 2: Weak
- 3: Adequate
- 4: Good
- 5: Excellent



- 24 test queries:
  - Ambiguous,
  - Misspelled
  - Vague
  - Partial
  - Contradictory
  - Rare entities
  - Multi-topic
- Used ‘debug_retrieval() to trace chunk relevance
- Final system score: 4.96 / 5 average

Debugging & Fixes

- Added document titles and section headers to chunks
- Used topic metadata for filtering and fallback search
- Improved prompts for definition, reasoning, and comparison
- Fixed issues with Fordism, Lean vs. Fordism, timelines

Final Evaluation Results

- 23 of 24 answers scored 5/5
- 1 answer scored 4/5 (short input)
- Zero Hallucinations
- Polished fallback and topic control

Project Assets

- `KPMG_Solution.ipynb` - main notebook
- `kpmg_rag.py` - main Python script (similar to notebook)
- `README.md` - steup & usage
- `requirements.txt` - dependecies
- Debug and evaluation modes included

Next Steps

Model Enhancements

- Add hybrid keyword + semantic search
- Consider re-ranking with GPT or other rerankers
- Explore retrieval tuning using rerankers or multi-vector search
- Use LangChain Agents for follow-up query handling

User Experience & Interaction

- Add visual outputs: charts, timelines
- Add streaming support for partial or live responses

Deployment & Integration

- Deploy as an API using FastAPI or Flask
- Apply metadata filters via CLI (e.g., data range, topic)
- Package as a pip-installable library for reuse in other teams

Feedback & Improvement

- Add Feedback collection and logging for continual improvement

Design Choices & Thought Process

Core Technology Choices

- Chose LangChain for modularity and ecosystem maturity
- Used Pinecone over FAISS due to hosted scalability and real-time performance
- Alternatives considered: Chroma for local vectors, FAISS for GPU performance, native OpenAI RAG without LangChain

Accuracy Optimization

- Optimized for accuracy by including section headers in chunks
- Applied semantic filters based on inferred topics
- Prompt tuning provided the biggest boost to relevance and reliability in edge cases

Design Choices & Thought Process

Smart Prompting Strategy

- Added prompt tailoring for different query types:
  - “why”, “how”, “define”, vauge, short, ambiguous, contradictory, etc.

Trade-offs Considered

- Accuracy vs. Speed:
  - Chose smarter prompts and richer context over minimal latency
- Cloud vs. Local vector DB:
  - Prioritized ease of deployment with Pinecone

Conclusion

- Built a modular and accurate RAG system using LangChain, OpenAI and Pinecone
- Designed to handle a wide range of queries with prompt eng. and topic filtering
- Achieved 96% success rate across 24 edge-case tests
- Clean, testable, and ready to scale or adapt
- Prepared for real-world usage: CLI, APP, frontend integrations, and team reuse (Will provided separately)
- Ready for further innovation and deployment

Thank you