Introduction

In this final report, I walk through my work on Tasks 1–3—document visualization, Gen AI extraction, and a GraphRAG chatbot for 10-K filings—highlighting my objectives, methods, experiments, and what I learned along the way. I'll also weave in more of my original narrative to capture the real challenges I faced.

Task 1 – Engineering: 2D Visualization & Outlier Detection

What I Did

- **Chunking (parsing.py & chunking.py)**
  Early on, I chased "perfect" chunking—trying to detect titles vs. body text via punctuation, newline characters, and word count features. Unfortunately, the filings were essentially HTML run through html2text, so I hit tons of edge cases. I even considered two fancy alternatives:

  1. **Embedding-based chunking (using semantic similarity to split)—but I realized it would depend heavily on the embedding model's precision and demand huge compute, so I scrapped it.**
  2. **LLM-based chunking (ask a large model to split)—but that felt like cheating since I'm building an LLM app, and it removed the engineering challenge.**
     In the end, I settled on my heuristic: split on newlines, classify short/no-punctuation lines as "non-normal" headings, then build groups by merging subsequent paragraphs until I hit a normal paragraph. This simple approach handled most cases and let me keep moving forward.

- **Embeddings & PCA**
  I embedded each chunk with Sentence-BERT—splitting any chunk over the model's token limit. I then standard-scaled the embeddings and plotted cumulative variance versus number of PCA components. When I saw ≥95 % variance at 200 components, I hard-set that as my hyperparameter.

- **Clustering & Outlier Flagging**
  I ran KMeans from 15 to 80 clusters, using the silhouette score to choose the best. (I picked 15 as a theoretical minimum—there are about 15 distinct sub-headings—and 80 because I guessed each section might split into up to four sub-clusters.) Beyond 80, the score crept up but felt meaningless, so I

stuck with that window. I flagged a chunk as an outlier if its distance to the nearest cluster centroid exceeded the 90th percentile across all chunks.

- **Visualization**
  For plotting, I used t-SNE to reduce the 200-dimensional PCA results to 2D. I generated scatter plots colored by cluster, by outlier flag, and by original section number. The visualizations looked clean, though I noted that t-SNE's compression sometimes warped true distances and made outliers less distinct.

Key Takeaway
I learned that a rule-based heuristic—though imperfect—can outperform over-engineered solutions when data quality is low. Chasing perfection on chunking would've stalled the project; a "good enough" method kept me productive.

Task 2 – Gen AI Extraction: Embedding-Based QA Pipeline

What I Did

- **Assumptions**
  I assumed users always specify:
  1. **A year (2018–2020)**
  2. **A company (Apple, Google, or Microsoft)**
  3. **A single factual metric (e.g., "number of weeks in fiscal year 2017")**
- **Filtering & Retrieval**
  I first filtered chunks by year and ticker. Then I embedded the user's query and each section description to pick the top 3 most relevant sections, shrinking ~2,800 chunks to ~100.
- **Summarization & Re-ranking**
  To reduce noise, I summarized each chunk to ≤200 tokens and re-embedded those summaries. From each of the 3 sections, I took the top 3 summaries (9 in total), then picked the 5 with highest similarity to the query.
- **LLM Prompting**
  I passed those 5 context chunks to GPT-4 with a structured prompt asking for a concise factual answer. I validated against a small ground-truth set (5 values per query).

Results

- **I achieved 5/5 correct answers for Apple and Microsoft.**
- **Two Google queries failed only because key tables never made it into the plain-text conversion (e.g., COVID-19 impact stats).**

Key Takeaway
Embedding-based section filtering plus chunk summarization gave me a fast, accurate pipeline without exhaustive search over thousands of chunks.

Task 3 – Chatbot Application: GraphRAG Pipeline

What I Did

- **After Task 2's filtering, I built a knowledge graph per sub-dataset using Llama-Index's GraphRAG (defaults: max 10 triplets/chunk, neighbor depth 3).**
- **I routed user queries through the GraphRAG API, letting it traverse nodes and edges to formulate answers.**
- **I compared accuracy, answer completeness, and response time against my embedding pipeline.**

Results

- **GraphRAG matched the 5/5 accuracy with virtually no extra tuning.**
- **Answers felt more coherent and contextually linked, thanks to the graph structure.**
- **But runtime ballooned to ~2.5 minutes per question, versus seconds for the embedding approach.**

Key Takeaway
Graph reasoning delivers richer answers at the cost of speed. I'll explore caching, parallel graph construction, or lighter graph pruning to speed things up.

Overall Conclusions & Next Steps

1. **Embrace "Good Enough"**
   I spent too long on perfect chunking before realizing a simple heuristic was more productive. Next time, I'll balance rigor with pragmatism.

2. **Optimize GraphRAG**
   I'll tune triplet limits, prune low-value nodes, and batch requests to cut down latency.
3. **Expand & Automate**
   I plan to broaden the query set, include more companies and filing types, and automate hyperparameter selection (chunk size, section thresholds, graph depth) based on query complexity.

Personal Reflections

I learned that it's all too easy to get stuck on edge cases—like bullet-list paragraphs—and lose sight of progress. Task 2 showed me how embeddings can zero in on relevant content almost instantly, while Task 3 proved that graph-based reasoning can elevate answer quality, even if it's slower. Most importantly, I'll carry forward the lesson to iterate fast, accept "good enough" where appropriate, and always measure real-world performance. I'm excited to tackle the next round of optimizations!