Architecture & Agent Design Report

Objective: Design an intelligent agent that uses GCP-native NLP tools to answer complex business queries from unstructured customer feedback documents.

Agent Goal: Empower business users to give feedback or product reviews.

Tools Used:

- Vertex AI: For generative summarization using Gemini models.
- Cloud Natural Language API: For entity extraction and sentiment analysis.
- Cloud Storage: To store and retrieve input text files.
- Vertex AI Pipelines: To orchestrate summarization and extraction processes.
- Artifact Registry: To host custom Docker images for component execution.

Explanation of the agentic workflow:

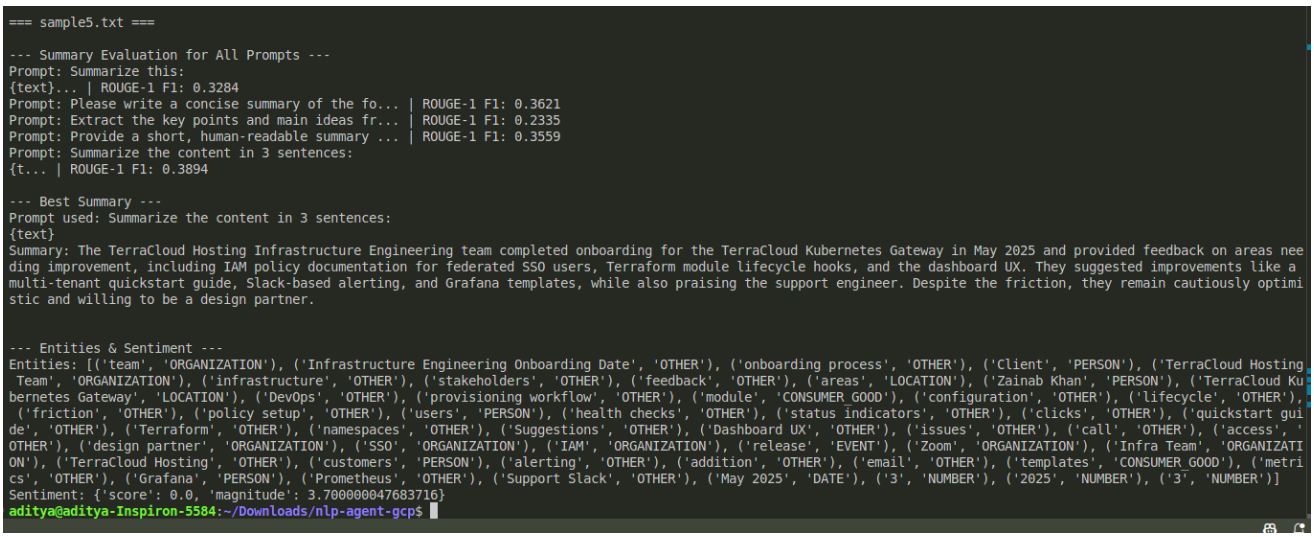
1. Load recent documents from GCS

2. Preprocess and clean text

3. For each doc, I am generating 5 different summaries based on 5 different summarization prompts and comparing them based on rouge scores. Displaying the summaries and rouge scores for each prompt. And eventually choosing the best summary and prompt.

4. After that I'm also extracting the entities and sentiments from the original docs using GCP Natural language API.

Below is a sample output:



The sentiment score from GCP NLP API can be between -1 (very negative) to +1 (very positive), The other metric called Magnitude reflects the intensity of emotion. Like in the below example indicates very negative feedback, this is the output of sample1.txt

Sentiment: {**'score'**: -0.5, **'magnitude'**: 6.800000190734863}

Reasoning logic Scenario (pseudocode and request flow):

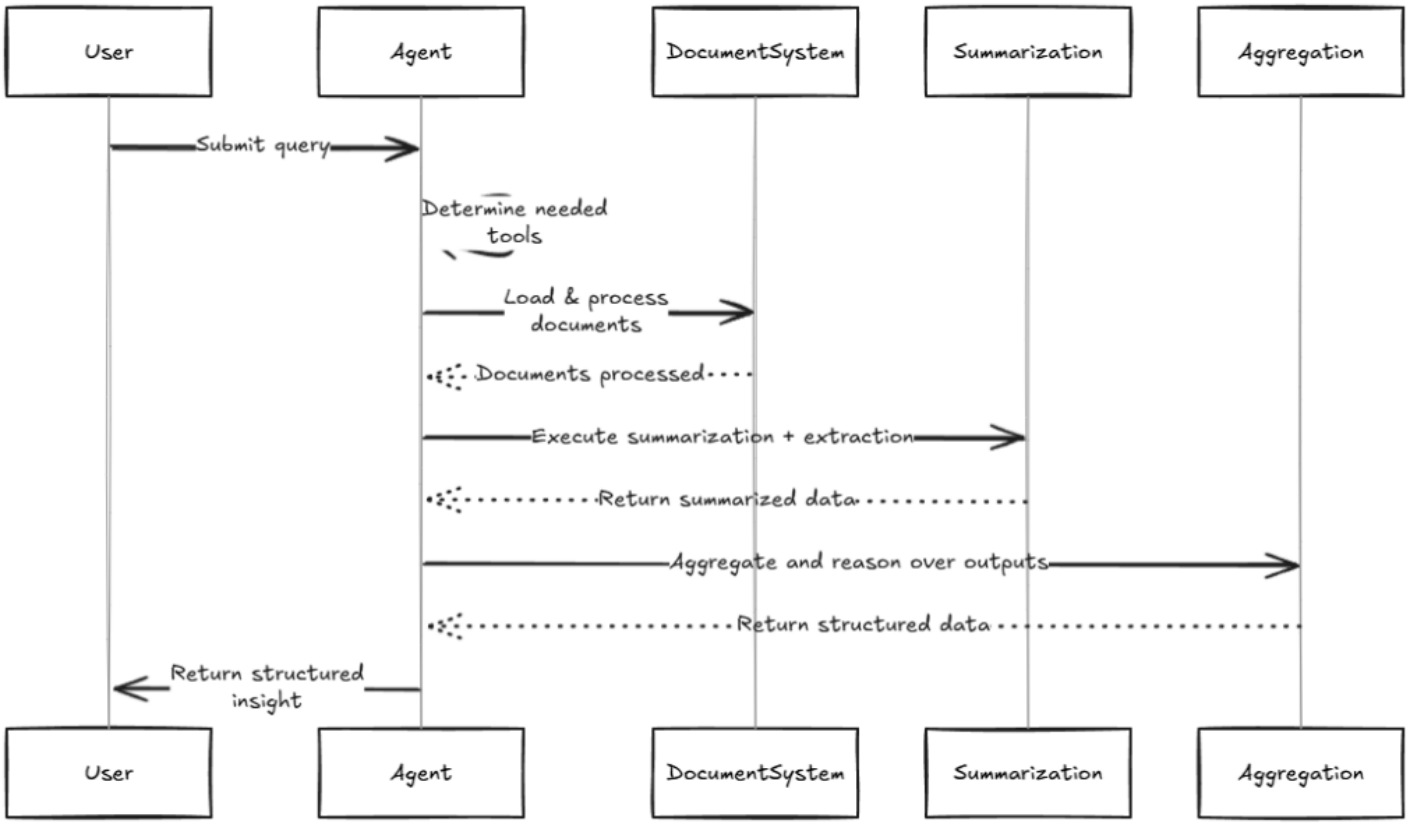
Below is the pseudocode logic and high-level workflow for tweaking my agent for the **Customer Insight Agent scenario**

Pseudocode:

```
def customer_insight_agent(user_query):
  docs = load_documents(gcs_prefix='docs/')
  results = []
  for doc in docs:
    cleaned = clean_text(doc)
    summary = generate_summary(cleaned)
    insights = extract_entities_sentiment(cleaned)
    results.append({ 'summary': summary, **insights })
  if 'negative' in user_query:
    filtered = [r for r in results if r['sentiment']['score'] < 0]
    top_issues = count_entity_mentions(filtered)
    return compile_answer(user_query, top_issues, filtered)
```

It begins by loading documents from a GCS bucket, then cleans each document, summarizes it using a generative model, and extracts named entities and sentiment using NLP tools. If the user query involves negative feedback, it filters the results for documents with negative sentiment, identifies the most frequently mentioned issues, and compiles a structured response. This design enables the agent to perform multi-step reasoning and deliver actionable insights from unstructured text

Flow diagram:



(Optional) Memory: Use BigQuery to persist structured entity/sentiment records. Use Firestore to store session-level context. Enable query reuse or caching via session tokens.

Discussion of results, challenges, and trade-offs.

1. **Entity and Sentiment extraction**
   Cloud Natural Language API effectively extracted entities and was able to generate sentiment scores for customer reviews. We got fast and scalable results without training a custom model.

2. **Summarization**
   Vertex AI (Gemini) generated concise summaries with reasonable accuracy. Prompt variations yielded more informative outputs depending on query type

Challenges and Trade Offs

Manual or heuristic review was needed for quality assessment. Vertex AI models are powerful but more costly and less tunable than custom models. GCP NLP API is accurate and easy to use, but less customizable for domain-specific entities. ROUGE scores provide rough benchmarks but miss nuance in abstractive summaries.

Productionization Strategy

- **Scalability and Orchestration**
  Vertex AI Pipelines to orchestrate multi-step NLP workflows, we can use real-time processing with Cloud Functions on GCS file upload events, Pub/Sub to decouple document ingestion from processing.

- **Security and Data Privacy**
  We can enforce IAM with least privilege roles (Vertex AI User, Storage Viewer, etc.). We can have Customer Managed Encryption Keys (CMEK) for GCS, BigQuery, and Firestore.

- **Monitoring, Logging, and Error Handling**
  We can Use Cloud Logging to trace component-level failures and summaries, dashboards for latency, error rates, and document throughput. For error handling, retries and timeouts to external API calls (e.g., Vertex AI, NLP API)

- **Cost and Optimization**
  Implement batching and document-level parallelism for efficiency, cache extracted entities/summaries in BigQuery or Firestore to prevent recomputation.

- **CI/CD**
  Use GitHub Actions or Cloud Build to automate Docker image build and push to Artifact Registry. Version pipeline definitions and test runs before promoting to production. Apply deployment policies with staged rollouts and manual approvals.