SCRIBE v3 Objectives

Phase 1: Foundation and Minimum Viable Product (MVP)

This phase focuses on setting up the core infrastructure and building the simplest end-to-end version of the project to prove the concept.

Objective 1: Establish the Google Cloud Environment
The first step is to prepare your cloud workspace. This involves creating the project, enabling the necessary services, and configuring your local command-line tools for interaction.

- **Key Tasks:**
  - Create a new Google Cloud Platform (GCP) project.
  - Enable billing for the project.
  - Using the gcloud CLI, enable the required APIs: Vertex AI, Cloud Storage, Cloud Run, Document AI, and BigQuery.[1]
  - Install and configure the Gemini CLI on your local machine to interact with the Gemini models.
  - Set up a service account with the appropriate IAM roles (e.g., Vertex AI User, Storage Object Admin) to manage resources programmatically.[1]

Objective 2: Implement the Project Gutenberg Ingestion Pipeline
This objective focuses on acquiring and processing the first set of source materials from Project Gutenberg.

- **Key Tasks:**
  - Write a script (in Python, for example) to download public domain books. Use a Gutenberg mirror site and respect roboting guidelines, such as using wget with a wait timer.[2]
  - Create a Google Cloud Storage (GCS) bucket to serve as the "data lake" for raw text files.
  - Upload the downloaded books to the GCS bucket.
  - Develop a simple text-cleaning function, perhaps in a Cloud Function triggered by new files in GCS. This function should remove the standard Project Gutenberg headers and footers to isolate the core text.[5]

Objective 3: Build the Core Semantic Processing Workflow
This is the heart of the RAG system, where text is converted into a searchable format.

- **Key Tasks:**
  - Develop a "chunking" strategy. Start with a content-aware approach, like splitting texts by paragraph or using a recursive character splitter.[5] The goal is to create chunks that are semantically coherent.

    - Use the Gemini API to generate vector embeddings for each text chunk.[6] When embedding these documents for storage, specify the taskType: 'RETRIEVAL_DOCUMENT' to optimize them for search.[7]
    - Set up a Vertex AI Vector Search index.[8]
    - Write the script to populate the index with the embeddings and basic metadata (like the source document ID and chunk text).

Objective 4: Develop the MVP Conversational Chat Assistant
This objective delivers the first interactive feature, allowing you to test the end-to-end flow.

- **Key Tasks:**
    - Create a simple backend application using a framework like Flask or FastAPI and prepare it for deployment on Cloud Run.
    - This backend will accept a user's text query.
    - Embed the incoming user query using the same Gemini model, but this time specifying taskType: 'QUESTION_ANSWERING' for optimal retrieval.[7]
    - Perform a similarity search against your Vertex AI Vector Search index to find the most relevant text chunks.
    - Engineer a basic prompt that combines the user's question with the retrieved text chunks.
    - Send this prompt to a Gemini Pro model to generate a grounded answer.
    - Return the generated answer.

Phase 2: Corpus Expansion and Feature Enrichment

With the core system working, this phase expands the knowledge base and refines the system's capabilities.

Objective 5: Implement the Sefaria Ingestion Pipeline
This objective adds a highly structured and valuable data source, requiring a specialized approach.

- **Key Tasks:**
    - Use the Sefaria API to programmatically fetch texts. It is critical to retrieve the full JSON object for each text, not just the plain text, as this contains invaluable structured metadata like canonical references (ref), section names, and intertextual links.[9]
    - Develop a dedicated parser (e.g., in a Cloud Function) that extracts information from the Sefaria JSON response.
    - Use the text's inherent structure (e.g., verse, daf, mishna) as the basis for your chunking strategy.
    - Map the rich Sefaria metadata to your internal schema and store it alongside the

vector embeddings in Vertex AI Vector Search.

Objective 6: Implement the Full Metadata Schema
This objective transforms the system from a simple text search into a scholarly tool by adding deep contextual information.

- **Key Tasks:**
  - Formally define the complete metadata schema outlined in the project plan (including fields like canonical_reference, author, publication_date, text_type, and relational links).
  - Modify all existing ingestion pipelines (Gutenberg, Sefaria, etc.) to extract or generate this metadata for each chunk.
  - For fields like topics_keywords or historical_context, you can make a call to a Gemini model within the pipeline to generate this information automatically.
  - Ensure this rich metadata is stored as a payload with each vector in Vertex AI Vector Search to enable powerful filtered queries.

Phase 3: Advanced Features and Full Release

This phase focuses on building the "Deep Research" functionality and preparing for a full public release.

Objective 7: Build the "Deep Research" Multi-Step RAG Feature
This objective creates the project's most powerful feature, capable of synthesizing information on complex topics.

- **Key Tasks:**
  - **Query Decomposition:** For a complex user topic, make an initial call to a Gemini model with a prompt instructing it to break the topic into a series of logical sub-questions.[11]
  - **Iterative Retrieval:** Create a loop that executes a standard RAG query (as developed in Objective 4) for each sub-question. Collect the results from each step.[14]
  - **Final Synthesis:** Once the loop is complete, aggregate all the retrieved context into a single, large prompt. Use a sophisticated prompt that instructs the Gemini model to act as a scholar and synthesize the materials into a coherent, well-structured essay on the original topic.[15]

Objective 8: Implement Advanced Prompt Engineering and Dialogue Management
This objective focuses on refining the quality and naturalness of the AI's interactions.

- **Key Tasks:**
  - Create a "prompt library" with distinct, optimized prompts for each major task (e.g., conversational chat, query decomposition, final synthesis).[17]

- Implement persona-based generation by defining SCRIBE's role and constraints in a master system prompt.[15]
- For the chat feature, implement conversation history. When a follow-up question is asked, use a preliminary LLM call to combine the history and the new question into a single, self-contained query before sending it to the RAG pipeline.[11]

Phase 4: Governance and Ongoing Optimization

This final set of objectives ensures the project is sustainable, cost-effective, and continuously improving.

Objective 9: Establish Cost Management and Governance
A crucial step for ensuring the project's long-term financial viability.

- **Key Tasks:**
  - Set up GCP billing data export to BigQuery for detailed analysis.[20]
  - Create budgets and spending alerts in the Google Cloud console to prevent unexpected costs.[23]
  - Apply labels to all GCP resources (e.g., component:ingestion, feature:chat) to enable granular cost tracking and attribution.[20]

Objective 10: Create an Evaluation Framework
To measure performance and guide improvements, you need a consistent way to test the system.

- **Key Tasks:**
  - Create a "golden dataset" consisting of a representative set of questions with ideal, human-approved answers.
  - Write an evaluation script that runs the questions from your golden dataset against the system and logs the generated answers.
  - Use an LLM-as-a-judge approach, where you prompt a powerful model like Gemini Pro to compare the system's output against the golden answer and score it on metrics like faithfulness and relevance. This provides a quantitative way to track quality over time.