Project SCRIBE: A High-Level Implementation Plan

Section 1: Foundational Architecture for Project SCRIBE on Google Cloud

The successful implementation of Project SCRIBE hinges upon a foundational architecture that is both powerful enough to handle the semantic complexity of its corpus and flexible enough to accommodate its unique feature set. The system's core, built on Retrieval-Augmented Generation (RAG), requires a deliberate choice of implementation model on the Google Cloud Platform (GCP). This choice dictates the balance between development velocity, operational overhead, and the granular control necessary to achieve the project's ambitious goals. An analysis of available patterns reveals that a hybrid approach, leveraging the strengths of custom logic and managed services, presents the optimal path forward.

1.1 Analysis of RAG Implementation Models on GCP

GCP offers a spectrum of services for building RAG applications, which can be categorized into three primary architectural patterns. Each presents a different set of trade-offs regarding control, speed, and complexity.

1. **Fully Managed (Vertex AI Search & Conversation):** This model represents the highest level of abstraction. Services like Vertex AI Search & Conversation (formerly known as Agent Builder) provide an end-to-end solution that can ingest documents from sources like Google Cloud Storage, automatically create embeddings, manage the vector database, and serve a conversational interface. While this approach offers the fastest possible time-to-market and minimal infrastructure management, it provides the least control over the critical stages of the RAG pipeline. The parsing, chunking, and metadata extraction processes are largely opaque and standardized, making this model unsuitable for the highly specialized and diverse corpus of Project SCRIBE.
2. **Partly Managed (Vertex AI RAG Engine):** Occupying a middle ground, the Vertex AI RAG Engine offers a managed orchestration service that streamlines the complex process of connecting data sources to an LLM. It handles the mechanics of data retrieval, chunking, embedding, and LLM integration, freeing developers from significant infrastructure management. This model provides a "sweet spot" for many applications, offering a balance of ease of use and customization. However, for a project requiring bespoke parsing logic for sources like the Sefaria API and nuanced, content-aware chunking for theological texts, the RAG Engine's level of control may still prove insufficient.
3. **Full Control (DIY RAG):** This pattern provides maximum flexibility by allowing developers to manually orchestrate individual GCP components to build a custom RAG pipeline. In this model, the developer is responsible for every step: processing documents with tools like Document AI, implementing custom chunking logic, generating embeddings with the Text Embedding API, managing a vector database with Vertex AI Vector Search, and

constructing the final prompt for the Gemini LLM. While this approach requires the most significant development effort, it offers the complete control necessary to address the unique challenges posed by Project SCRIBE's source material and feature requirements.

The very nature of Project SCRIBE—its focus on "deep meaning," its reliance on ancient texts with complex structures, and its need to integrate rich, scholarly metadata—precludes the use of a one-size-fits-all managed solution. The value of the system will be derived not from a generic RAG implementation but from a highly tailored one. Therefore, the "Full Control" model is the only viable path to achieving the project's vision.

1.2 Recommended Hybrid "Full Control" Architecture for SCRIBE

While the "Full Control" pattern is necessary, it does not imply that every component must be built from scratch. A purely "do-it-yourself" approach, such as managing an open-source vector database like FAISS on a Google Compute Engine (GCE) virtual machine, would introduce significant and unnecessary operational complexity. The most strategic path forward is a hybrid architecture that combines the "Full Control" application logic with the power of managed GCP infrastructure components. This provides the required customization for domain-specific tasks while offloading the undifferentiated heavy lifting of infrastructure management to Google's scalable, production-ready services.

The recommended architecture is as follows:

- **Data Ingestion & Preprocessing:** A custom, event-driven pipeline will form the ingestion layer. Google Cloud Storage (GCS) will serve as the central data lake for all raw source material. The arrival of a new document in a GCS bucket will trigger a Cloud Function or Cloud Run service. This service will orchestrate the preprocessing, using Google's Document AI for its powerful Optical Character Recognition (OCR) and text extraction capabilities on any PDF or scanned image sources.
- **Embedding & Indexing:** A dedicated microservice, hosted on Cloud Run for scalability, will be responsible for the semantic core of the system. It will implement the project's custom, content-aware chunking strategies. For each chunk, it will call the Gemini Text Embedding API (e.g., text-embedding-004) to generate high-quality vector embeddings. These embeddings, along with their rich metadata, will be stored and indexed in Vertex AI Vector Search.
- **Retrieval & Generation:** The main application backend, also hosted on Cloud Run, will handle user interactions. When a user submits a query, the backend will embed it using the same Gemini model, perform a multi-stage retrieval against Vertex AI Vector Search, construct a sophisticated, context-rich prompt, and call the appropriate Gemini Pro model (e.g., Gemini 1.5 Pro) on Vertex AI for the final reasoning and text generation.
- **Metadata & Logging:** Structured metadata, which is critical for the project's success, will be stored alongside the vectors in Vertex AI Vector Search to enable filtered queries. A copy of this metadata, along with operational logs, will be exported to BigQuery. This enables large-scale analytics, cost tracking, and the evaluation of system performance over time.

This hybrid architecture represents a strategic choice. It concentrates development effort on the areas that create unique value for SCRIBE—the nuanced parsing of ancient texts, the hierarchical chunking strategies, and the design of a rich metadata schema. Simultaneously, it leverages Google's expertise in managing massive, low-latency systems for vector search and model serving, ensuring the final product is both intelligent and enterprise-grade. The selection of Vertex AI Vector Search is particularly critical. While other GCP databases like AlloyDB and

Cloud SQL now offer vector capabilities, a dedicated service like Vertex AI Vector Search is purpose-built for the high-performance, metadata-filtered similarity searches that are fundamental to SCRIBE's context-rich retrieval process. This choice future-proofs the system's ability to integrate metadata deeply into the retrieval loop.

Architectural Pattern	**Key Services**	**Control over Ingestion**	**Control over Retrieval/Prompting**	**Development Velocity**	**Suitability for SCRIBE**
Fully Managed	Vertex AI Search & Conversation	Low	Low	Fast	**Low:** Unsuitable for the project's complex, non-standard corpus (e.g., Sefaria's structured data, biblical verse structure). Lacks the necessary control over chunking and metadata to enable "deep meaning."
Partly Managed	Vertex AI RAG Engine, Gemini, GCS	Medium	Medium	Medium	**Medium:** A strong option for standard enterprise RAG but lacks the fine-grained control over parsing and hierarchical chunking required for SCRIBE's specialized sources and advanced "Deep Research" feature.
Hybrid "Full Control" (Recommended)	Cloud Run, Document AI, Vertex AI Vector Search, Vertex AI	High	High	Slow	**High:** Provides the essential, granular control over the entire data pipeline,

Architectural Pattern	Key Services	Control over Ingestion	Control over Retrieval/Prompting	Development Velocity	Suitability for SCRIBE
	Gemini API, GCS, BigQuery				from source-specific parsing to custom hierarchical chunking and rich metadata integration. This flexibility is a prerequisite for achieving the project's goals, while managed components (Vector Search, Gemini API) ensure scalability and reliability.
Table 1.1: GCP RAG Architecture Decision Matrix for Project SCRIBE					

1.3 Core Service Integration Diagram

The flow of data and control within the proposed architecture is visualized as a series of interconnected, scalable services. The user-facing application communicates with a backend hosted on Cloud Run. This backend orchestrates the entire RAG process, interacting with the various Vertex AI services for embedding, retrieval, and generation. The ingestion pipeline operates asynchronously, triggered by events in Google Cloud Storage, ensuring the main application remains responsive.

```
graph TD
    subgraph User Interaction
        User[User Interface] --> AppBackend
    end

    subgraph Data Ingestion & Processing (Asynchronous)
        DS_Gutenberg[Project Gutenberg] --> IngestionGCS
        DS_Archive[Archive.org] --> IngestionGCS
        DS_Sefaria --> IngestionGCS
        DS_PDF --> IngestionGCS
```

```
        IngestionGCS -- triggers --> IngestionPipeline
        IngestionPipeline -- uses --> DocAI
        IngestionPipeline -- chunks & embeds --> EmbeddingService
        EmbeddingService -- calls --> GeminiEmbeddingAPI[Vertex AI:
Gemini Embedding API]
        EmbeddingService -- writes to --> VectorSearch
        EmbeddingService -- writes to --> BigQueryMeta
    end

    subgraph RAG Inference Pipeline (Synchronous)
        AppBackend -- embeds query --> GeminiEmbeddingAPI
        AppBackend -- retrieves from --> VectorSearch
        AppBackend -- constructs prompt & generates -->
GeminiGenerationAPI[Vertex AI: Gemini Pro API]
        GeminiGenerationAPI --> AppBackend
        AppBackend --> User
    end

    style User fill:#d4e1f5,stroke:#333,stroke-width:2px
    style DS_Gutenberg fill:#f5d4d4,stroke:#333,stroke-width:1px
    style DS_Archive fill:#f5d4d4,stroke:#333,stroke-width:1px
    style DS_Sefaria fill:#f5d4d4,stroke:#333,stroke-width:1px
    style DS_PDF fill:#f5d4d4,stroke:#333,stroke-width:1px
```

1.4 Scalability, Security, and Reliability

The proposed architecture is inherently designed for the demands of a production environment, addressing scalability, security, and reliability as first-class concerns.

- **Scalability:** The use of serverless and managed services is key to elastic scalability. Cloud Run, Cloud Functions, and the Vertex AI suite are designed to scale automatically based on traffic. This means the system can handle large, bursty data ingestion jobs and scale to accommodate a growing number of concurrent users without manual intervention. The architecture avoids performance bottlenecks by decoupling the synchronous user-facing pipeline from the asynchronous data ingestion pipeline.
- **Security:** A multi-layered security posture will be implemented. Access to all GCP resources will be governed by the principle of least privilege using fine-grained Identity and Access Management (IAM) roles. All data at rest within Google Cloud Storage, Vertex AI Vector Search, and BigQuery will be protected using customer-managed encryption keys (CMEK), providing an additional layer of control over data security. Furthermore, Google Cloud's Binary Authorization will be used to ensure that only trusted and verified container images are deployed to the Cloud Run environments, mitigating the risk of deploying compromised code.
- **Reliability:** High availability is achieved by leveraging Google's global infrastructure. Key services like Google Cloud Storage and Vertex AI endpoints can be configured for multi-regional availability, ensuring resilience against zone-level or even regional outages. The serverless nature of Cloud Run also contributes to reliability, as Google manages the

underlying infrastructure, including automatic failover and load balancing.

Section 2: The Corpus: A Multi-Source Data Ingestion and Management Strategy

The knowledge base, or corpus, is the lifeblood of Project SCRIBE. Its quality, diversity, and structural integrity will directly determine the depth and accuracy of the insights the system can provide. The ingestion process cannot be a monolithic, one-size-fits-all pipeline. Instead, it must be a sophisticated framework of source-aware processes designed to acquire, clean, validate, and maintain data from a variety of unique sources. This requires bespoke pipelines that respect and preserve the inherent structure of each data source, particularly the rich, linked data from Sefaria.

2.1 Phase 1: Data Acquisition Framework

A systematic and respectful data acquisition strategy is the first step. This involves using the appropriate tools and methods for each source, adhering to their terms of service and best practices.

- **Project Gutenberg:** For bulk acquisition of its vast library of public domain works, we will primarily utilize the official mirror sites. This is a key guideline from Project Gutenberg to prevent overloading their main servers. Automated tools like wget with appropriate rate limiting (e.g., a 2-second wait between requests) or dedicated bulk downloaders can be employed for this purpose. Programmatic discovery of works will be driven by the machine-readable XML/RDF catalog, which provides comprehensive metadata for the entire collection. This metadata is crucial for populating our own schema during ingestion. For more dynamic or targeted lookups, APIs like Gutendex, which provide a clean JSON interface to the catalog, can be used as a supplementary tool. The acquisition process will prioritize plain text (.txt) and HTML formats, as these are the most straightforward to parse and process.
- **Archive.org:** The acquisition of materials from Archive.org will be handled programmatically to ensure scalability and reproducibility. We will utilize an API client, such as the internetarchive R package or a Python equivalent, which allows for advanced, metadata-based searching. This enables us to target specific collections, publishers, or date ranges, rather than performing blind keyword searches. The primary goal will be to download full-text files, typically in .txt format. For more granular access, such as retrieving specific versions of archived web pages, the CDX/C API can be leveraged. All interactions with the Archive.org APIs will be conducted with caution, respecting any stated rate limits to ensure responsible use.
- **Sefaria.org:** This source is arguably the most valuable for Project SCRIBE due to its highly structured and interconnected data. We will use the official Sefaria REST API, which is publicly available and does not require an API key for read access. The critical aspect of this acquisition process is to retrieve the full JSON object for each text reference, not just the text itself. This JSON response contains a wealth of structured information that is essential for our project, including:
    - ref: The canonical, citable reference (e.g., "Genesis 1:1").
    - sectionNames: The structural hierarchy of the text (e.g., ["Chapter", "Verse"] for the Bible, or for Mishneh Torah).

    - commentary and links: Arrays that explicitly define intertextual connections to other works in the Sefaria library. This is a pre-built knowledge graph that must be preserved.
- **Other Public Domain Sources:** For any additional materials not available through these primary channels (e.g., specific academic papers, out-of-copyright scholarly books), a semi-automated workflow will be established. These documents will be manually uploaded to a dedicated "ingestion" bucket in Google Cloud Storage, from where they will enter the standard processing pipeline.

2.2 Phase 2: Source-Specific Processing Pipelines

The core of the ingestion strategy lies in recognizing that each source provides data in a different format and with a different level of structure. A single processing pipeline would destroy this valuable information. Therefore, we will develop a set of distinct, source-aware micro-pipelines, likely implemented as independent Cloud Functions or Cloud Run services.

- **Gutenberg/Archive.org Pipeline:**
    - **Parsing:** This pipeline focuses on extracting clean text from semi-structured or unstructured files. For HTML sources, it will strip all tags to get the raw text content. For plain text files, it will normalize character encoding (to UTF-8) and handle inconsistent line breaks.
    - **Cleaning:** A significant challenge with sources like Project Gutenberg is the presence of boilerplate headers and footers that describe the project, licensing, etc.. The pipeline will include a cleaning step that uses a combination of rule-based logic (e.g., looking for keywords like "Project Gutenberg") and potentially a simple machine learning classifier to identify and excise these non-content sections.
    - **Structuring:** The pipeline will attempt to infer the document's structure (e.g., chapter and paragraph breaks) from visual cues in the plain text, such as multiple newlines, indented lines, or capitalized headings.
- **Sefaria Pipeline:**
    - This pipeline's primary task is not text extraction but the meticulous parsing of the JSON response from the Sefaria API. The goal is to translate the rich structure of the Sefaria object directly into our internal metadata schema (defined in Section 4).
    - The text content (in both Hebrew and English) will be extracted, but more importantly, the metadata fields like ref, sectionNames, categories, and the links array will be preserved as first-class data. This ensures that the invaluable intertextual relationships curated by Sefaria are captured and integrated into our system's knowledge graph.
- **PDF/Scanned Document Pipeline:**
    - Many valuable historical and philosophical texts may only be available as PDFs or scanned images of physical books. For these sources, the pipeline will rely heavily on **Google's Document AI**. Document AI's advanced OCR capabilities are essential for extracting high-fidelity, accurate text from these challenging formats. The quality of the OCR output is a direct determinant of the quality of the final embeddings, making this a critical component for corpus expansion.

2.3 Phase 3: Data Validation and Quality Assurance (QA)

The unique challenges of working with ancient and historical texts—such as fragmented

sources, damaged manuscripts leading to poor scans, and non-standard digitization practices—necessitate a robust quality assurance process. A purely automated pipeline risks ingesting erroneous data, which would undermine the project's goal of providing deep and accurate insights. An error in a key theological term or a misattributed philosophical passage could have significant downstream consequences. This reality demands a "Human-in-the-Loop" approach to data quality.

- **Automated QA:** The ingestion pipelines will include automated checks to catch common, low-level errors. These scripts will validate file integrity, detect character encoding problems, and flag documents with potential OCR gibberish (e.g., by analyzing character frequency distributions or the ratio of non-dictionary words).
- **Manual QA Interface:** A simple, secure web interface will be developed for project staff and subject matter experts. This tool will present ingested texts, particularly high-value ones like the core biblical books and major philosophical works, for human review. Reviewers will be able to correct transcription errors, flag poorly parsed sections, and validate the automatically extracted metadata. This step is not an optional extra; it is a fundamental requirement for building a trustworthy knowledge base and necessitates budgeting for the time of data curators or domain experts.
- **Source Provenance:** To ensure full traceability and maintain data integrity, every piece of ingested content will be tagged with comprehensive provenance metadata. This includes the original source (e.g., "Project Gutenberg," "Sefaria"), the specific item ID or URL, the date of acquisition, and the version of the ingestion pipeline used to process it.

2.4 Corpus Update and Maintenance Strategy

The corpus of Project SCRIBE will not be static. New books will be added to public domain archives, and sources like Sefaria may issue corrections or add new links. The system must be able to handle these updates efficiently and cost-effectively. We will adopt a hybrid update strategy inspired by advanced RAG architectures.

- **Incremental Updates:** For dynamic sources like Sefaria, an automated, trigger-based update process will be implemented. For example, a scheduled weekly Cloud Function will query the Sefaria API for any changes or additions and ingest only the new or modified data.
- **Selective Re-indexing:** A crucial cost-saving measure is to avoid re-processing the entire corpus for every small change. When a single document is updated, the pipeline will be designed to re-chunk and re-embed only that specific document and its associated data in the vector store. This selective re-indexing approach saves immense computational resources and time compared to a full rebuild.
- **Batch Processing:** For large-scale additions, such as ingesting a new collection of several hundred books, the changes will be grouped and processed together as a single batch job. This optimizes resource utilization by reducing the overhead associated with starting and stopping numerous small processing tasks.

Section 3: The Semantic Core: Advanced Chunking and Embedding with Gemini

The transformation of raw, ingested text into a semantically searchable knowledge base is the most critical technical step in the RAG pipeline. This "semantic core" determines the precision of

retrieval and the contextual richness available for generation. Project SCRIBE will reject a simplistic, fixed-size chunking strategy, which is ill-suited for its diverse and structured corpus. Instead, it will employ a sophisticated, multi-strategy approach to chunking that respects the inherent structure of the texts, coupled with an optimized implementation of Google's Gemini embedding models.

3.1 A Multi-Strategy Approach to Chunking

The guiding principle for chunking will be semantic coherence: a chunk must represent a complete, understandable unit of thought. A one-size-fits-all approach would fracture verses, break paragraphs mid-thought, and destroy the logical flow of the source material. Therefore, a content-aware chunking strategy is required.

- **Structural Chunking:** This will be the primary and preferred method, leveraging the natural divisions within the texts.
    - **For the Bible:** The fundamental unit of chunking will be the **verse**. This is the most granular, universally recognized, and citable unit of the biblical text. However, relying solely on verse-level chunks would lead to a "lost-in-the-middle" problem, where the system retrieves isolated sentences without their surrounding narrative context. To solve this, a **hierarchical chunking** strategy will be implemented. While individual verses will be the primary retrievable units, they will be linked to parent chunks representing pericopes (thematic sections, often denoted by subheadings in modern translations) and chapters. This hierarchical structure is not merely an optimization; it is a prerequisite for the advanced "Deep Research" feature, as it enables the system to retrieve both a specific point and its broader context.
    - **For Philosophical/Historical Books:** The default chunking unit will be the **paragraph**. Paragraphs are designed to encapsulate a single, coherent idea, making them ideal semantic chunks. For sources provided in structured formats like HTML or Markdown, the pipeline will also use section and chapter headings to build a structural hierarchy, similar to the biblical texts.
    - **For Sefaria Texts:** The system will defer to the expert-curated structure provided by the Sefaria API. For a text like the Talmud, the chunking will align with its native divisions, such as the Daf (folio page), Amud (side of the page), and Mishna or Gemara segment. Preserving this structure is essential for maintaining the text's integrity.
- **Recursive Character Splitting:** In cases where a source text is a long, unstructured block of plain text (e.g., some older files from Project Gutenberg), a fallback strategy is necessary. The system will use a **recursive character splitter**, such as the one available in the LangChain framework. This method attempts to split text along a prioritized list of separators (e.g., \n\n for paragraphs, then \n for lines, then sentences), ensuring that the splits are as semantically meaningful as possible while adhering to a maximum chunk size.
- **Sentence Window Retrieval ("Small-to-Big"):** To achieve the best of both worlds—high retrieval precision and rich generation context—the final implementation will adopt a **sentence window retrieval** or **parent document retrieval** strategy. In this model, the system will embed smaller, more specific units (e.g., individual sentences or small groups of verses) to enable highly precise similarity searches. However, once the most relevant small chunk is identified, the system will retrieve its larger parent chunk (e.g., the full paragraph or the entire pericope) to provide the LLM with sufficient context for reasoning

and generation. This decouples the unit of retrieval from the unit of synthesis, optimizing both processes independently.

3.2 Implementing Gemini Embeddings: taskType Optimization

The quality of the vector embeddings is paramount. Project SCRIBE will leverage one of Google's state-of-the-art text embedding models, such as text-embedding-004 or its successors, accessed via the Vertex AI API. These models have demonstrated superior performance on complex, specialized domains like legal discovery, which serves as a strong analogue for the dense theological and philosophical language in SCRIBE's corpus.
A critical, yet often overlooked, feature of the Gemini embedding API is the taskType parameter. This parameter allows the model to be optimized for the specific role a piece of text will play in the RAG pipeline. Failing to use this feature would be a significant missed opportunity for a "free" boost in retrieval accuracy. The project will mandate an asymmetric embedding strategy:

- **For Ingestion:** All document chunks being added to the knowledge base will be embedded using taskType: 'RETRIEVAL_DOCUMENT'. This fine-tunes the resulting vectors to be effective targets in a large-scale search index.
- **For User Queries:** When a user submits a query, that query will be embedded using a different task type. For the conversational assistant, the most appropriate type is taskType: 'QUESTION_ANSWERING'. For more general searches, taskType: 'RETRIEVAL_QUERY' can be used. This creates an optimized semantic search space where query vectors are specifically designed to find the most relevant document vectors, leading to a significant improvement in retrieval quality.

Corpus Type	**Primary Chunking Unit**	**Hierarchical Parent(s)**	**Fallback Strategy**	**Gemini Embedding taskType**	**Key Metadata Captured**
Bible (e.g., KJV)	Verse	Pericope, Chapter	N/A	RETRIEVAL_DOCUMENT	Canonical Reference, Book, Chapter, Verse
Talmud (Sefaria)	Sefaria Segment (e.g., Mishna)	Daf, Amud, Tractate	N/A	RETRIEVAL_DOCUMENT	Canonical Reference, Speaker, Intertextual Links
Platonic Dialogues (Gutenberg)	Paragraph	Section, Book	Recursive Character Split	RETRIEVAL_DOCUMENT	Canonical Reference (Stephanus), Speaker
Scanned Theological Journal (Archive.org)	Paragraph	Section, Article	Recursive Character Split	RETRIEVAL_DOCUMENT	Author, Title, Journal, Publication Date
Table 3.1: Corpus-Specific Chunking and Embedding					

Corpus Type	Primary Chunking Unit	Hierarchical Parent(s)	Fallback Strategy	Gemini Embedding taskType	Key Metadata Captured
Strategy					

3.3 The Vector Database: Structuring and Indexing in Vertex AI Vector Search

The chosen vector database is Vertex AI Vector Search, a fully managed service that provides the low-latency, high-throughput retrieval necessary for a production-grade application. Its key features align perfectly with the project's needs:

- **Hybrid Search:** The index will not be a simple list of vectors. Each vector entry in Vertex AI Vector Search can be accompanied by a rich payload of structured metadata. This is a critical feature, as it enables powerful hybrid search queries that combine semantic similarity with metadata filtering (e.g., "find texts semantically similar to 'grace' WHERE author = 'Augustine'").
- **High-Performance Indexing:** The system will utilize an Approximate Nearest Neighbor (ANN) index, most likely based on the Hierarchical Navigable Small World (HNSW) algorithm, which is the industry standard for balancing search speed and accuracy on large-scale vector datasets.
- **Scalability and Maintenance:** As a managed service, Vertex AI Vector Search handles the complexities of sharding, replication, and scaling. It also supports incremental updates to the index, allowing for the efficient addition or modification of documents without requiring a full re-index of the entire corpus, which is essential for our maintenance strategy.

3.4 Handling Multilingual Texts and Original Languages

A unique requirement for Project SCRIBE is its handling of original languages alongside English translations. The strong multilingual capabilities of the Gemini embedding models make this feasible.

The ingestion process will create distinct but linked records for texts in their original language (Hebrew, Aramaic, Greek) and their English counterparts. The metadata for a chunk of an English translation will contain a unique identifier pointing to the corresponding chunk in the original language, and vice-versa. This explicit linking within the metadata enables powerful cross-lingual research capabilities. A user could ask a question in English about a theological concept, and the system could retrieve not only the relevant English passages but also the source text in the original Greek or Hebrew. This allows the final LLM to analyze the nuances of the original terms, providing a level of depth unattainable with a monolingual corpus.

Section 4: A Schema for Meaning: Designing a Rich Metadata Framework

The metadata schema is the informational scaffolding upon which Project SCRIBE's "deep meaning" capabilities are built. It elevates the system from a simple text search engine into a true scholarly research assistant. A well-designed schema provides the essential context—bibliographic, structural, semantic, and relational—that allows the Large Language

Model (LLM) to ground its responses in historical and theological reality. Without this context, the LLM would be interpreting ancient texts in a vacuum. Therefore, the metadata is not merely data *about* the text; it is an integral *part* of the context provided to the LLM during the generation process.

4.1 Principles of Metadata for Digital Humanities

The design of the SCRIBE schema will be guided by established principles for creating robust and sustainable metadata in digital humanities and library science :

- **Modularity & Extensibility:** The schema will be built around a core set of common elements, but it will be designed to be extensible. This allows the system to accommodate the unique properties of specific texts (e.g., the detailed structural information from Sefaria) without cluttering the schema for simpler texts (e.g., a basic book from Project Gutenberg). We will use namespaces (e.g., scribe:, dc:) to ensure clarity and prevent ambiguity when mixing elements from different standards.
- **Refinement:** The schema will support varying levels of granularity. This principle acknowledges that it is not always feasible or necessary to have the same level of detail for every item in the corpus. The system must be able to function with both simple and richly detailed metadata records.
- **Interoperability:** To ensure the data is understandable and potentially reusable in other contexts, the core of our schema will be based on a widely adopted standard. We will use the **Dublin Core Metadata Element Set** as our foundation, as it provides a simple yet effective vocabulary for describing resources. We will then extend this base with domain-specific fields tailored to the needs of Project SCRIBE.

4.2 Proposed Metadata Schema for Project SCRIBE

The following schema represents the definitive structure for the metadata associated with every chunk in the knowledge base. This metadata will be stored as a JSON payload alongside its corresponding vector in Vertex AI Vector Search and will also be ingested into BigQuery for offline analysis and reporting.

Field Name	Data Type	Description	Source	Example
Core Bibliographic (Dublin Core Based)				
id	String	A unique, system-generated identifier for the chunk.	System Generated	c7b8-a9d0-e1f2
source_id	String	Identifier for the parent document from which the chunk originates.	Ingestion Pipeline	pg:10 (for Gutenberg eBook #10)
title	String	The title of the work.	Gutenberg RDF, Sefaria API, Manual	"The King James Bible"

Field Name	Data Type	Description	Source	Example
author	Array	The author(s) or attributed author(s) of the work.	Gutenberg RDF, Sefaria API, Manual	["Paul the Apostle"]
publisher	String	Publisher information, where available.	Gutenberg RDF, Manual	"Robert Barker"
publication_date	Integer	The year of publication or composition (negative for BC).	Gutenberg RDF, Sefaria API, LLM-Generated	1611
source_url	String	A direct URL to the original source material.	Ingestion Pipeline	https://www.gutenberg.org/ebooks/10
language	String	The ISO 639-2 language code of the chunk's text.	Ingestion Pipeline	"en" or "grc" (Ancient Greek)
Structural & Semantic (SCRIBE Specific)				
canonical_reference	String	A standardized, human-readable, and citable reference.	Sefaria API, Ingestion Pipeline	"John 3:16" or "Plato.Republic.5.472a"
text_type	String	The genre or type of text.	Sefaria API, LLM-Generated, Manual	"Gospel", "Epistle", "Socratic Dialogue"
chunk_level	String	The level of this chunk within the document's hierarchy.	Ingestion Pipeline	"Verse", "Paragraph", "Chapter"
parent_chunk_id	String	The ID of the immediate parent chunk in the hierarchy.	Ingestion Pipeline	p6c7-b8a9-d0e1
original_language_id	String	The ID of the corresponding chunk in the original language.	Ingestion Pipeline	h5b6-c7a8-b9d0
topics_keywords	Array	A list of key topics, themes, or keywords present in the chunk.	LLM-Generated	["faith", "salvation", "covenant"]
Relational (SCRIBE Specific)				

Field Name	Data Type	Description	Source	Example
relation_intertextuality	Array	An array of canonical references to other texts quoted or alluded to.	Sefaria API, LLM-Generated	["Isaiah 53:7-8"]
relation_commentary	Array	An array of source IDs for texts that are commentaries on this chunk.	Sefaria API, LLM-Generated	["sefaria:commentary-on-genesis"]
relation_thematic	Array	An array of chunk IDs that are thematically similar.	LLM-Generated (Clustering)	["c8d9-b0c1-f3g4", "d9e0-c1d2-g4h5"]
historical_context	String	A brief description of the historical period and relevant context.	LLM-Generated, Manual Curation	"Written during the Roman occupation of Judea, amidst rising messianic expectations."
Table 4.1: The SCRIBE Metadata Schema				

4.3 Automated Metadata Extraction and Manual Curation

Populating this rich schema requires a two-pronged approach.

- **Automated Extraction:** The ingestion pipelines will be designed to populate as much metadata as possible automatically. The Project Gutenberg RDF catalog and the Sefaria API are invaluable, pre-existing sources for bibliographic and structural data. For fields that require semantic understanding, such as topics_keywords or identifying potential relation_intertextuality links in texts that lack them, a Gemini model will be integrated directly into the ingestion pipeline. This model will read each chunk and generate the relevant metadata as part of the processing flow.
- **Manual Curation:** Automated methods can only go so far. Fields like historical_context and validating complex relational links require deep scholarly knowledge. The "Human-in-the-Loop" QA interface described in Section 2 will also serve as a curation tool, allowing domain experts to review, correct, and enrich the metadata for high-value texts, ensuring the highest level of accuracy for the core corpus.

4.4 Leveraging Metadata for Filtered Search and Contextual Grounding

The metadata schema is not a passive data store; it is an active and essential component of the RAG process.

- **Filtered Search:** The retrieval backend will leverage the metadata to dramatically improve search relevance and efficiency. Before performing a computationally intensive vector search, the system can first apply a metadata filter to narrow down the search space. For example, a query like "What did Augustine write about free will after the year

412?" would first execute a filter against the index for author: "Augustine" and publication_date > 412. The subsequent semantic search for "free will" is then performed only on this much smaller, highly relevant subset of the data.

- **Contextual Grounding:** This is the most critical application of the metadata. The metadata associated with the retrieved chunks will be passed to the final LLM prompt along with the text itself. The prompt will be explicitly engineered to instruct the LLM to use this information. For example: *"Given the following context, answer the user's query. Context Text: '...'. Context Metadata: {Author: 'Plato', Work: 'The Republic', Date: 'c. 375 BC', Type: 'Socratic Dialogue'}. User Query: '...'."* This provides the LLM with the essential frame of reference needed to interpret the text correctly and generate a nuanced, historically-aware response.
- **Knowledge Graph Traversal:** The relational metadata fields (relation_intertextuality, relation_commentary) effectively create a latent knowledge graph within the vector database. This structure is what enables the system to perform complex, multi-hop reasoning that a standard vector search cannot. For a query like, "How did early church fathers interpret Isaiah 53?", the "Deep Research" feature can first retrieve the text of Isaiah 53, then perform a second retrieval step to find all documents whose metadata contains a relation_commentary link pointing to that specific passage. This transforms a simple search into a sophisticated research process, allowing the system to follow chains of influence and interpretation across the entire corpus.

Section 5: Implementing Core Features: Conversational Exploration and Deep Research

Project SCRIBE will offer two primary user-facing features, each designed to serve a different mode of inquiry. The "Conversational Chat" provides a direct, interactive way to ask questions and get grounded answers. The "Deep Research" generator provides a powerful tool for synthesizing complex themes across the entire corpus. These two features are not merely different user interfaces on the same backend; they require fundamentally different RAG architectures and orchestration logic to function effectively.

5.1 Architecture for the Conversational Chat Assistant

The Conversational Chat Assistant is designed for speed, accuracy, and interactivity. Its architecture will be based on a standard, highly optimized RAG loop that prioritizes low latency for a fluid user experience.

- **Workflow:**
    1. **Query Submission:** A user enters a question into the web-based chat interface (e.g., "What is the Parable of the Sower?").
    2. **Query Embedding:** The application backend receives the query and embeds it using the Gemini embedding model, specifying the taskType: 'QUESTION_ANSWERING' for optimal relevance.
    3. **Single-Step Retrieval:** A single, efficient retrieval request is made to the Vertex AI Vector Search index. This request will include the query vector and may be augmented with metadata filters if the user's query implies them (e.g., "...in the Gospel of Mark"). The search returns the top-k most semantically similar chunks.
    4. **Prompt Construction:** The text and metadata from the retrieved chunks are

formatted into a concise prompt for the generation model. The prompt is designed for direct question answering.

5. **Generation and Citation:** The prompt is sent to a Gemini Pro model. The model generates a direct answer to the user's question, grounded in the provided context. The system simultaneously extracts the canonical_reference from the metadata of the source chunks.
6. **Response Delivery:** The generated answer, along with clear, clickable source citations, is streamed back to the user's interface, ensuring the response is both informative and verifiable.

This architecture is tailored for the request-response nature of a chat application, where users expect quick and factual answers to specific questions.

5.2 Architecture for the "Deep Research" Generator (Multi-Step/Recursive RAG)

The "Deep Research" feature is designed to tackle complex, multifaceted, and open-ended research topics that cannot be answered with a single retrieval step. This requires a more advanced, asynchronous, and iterative RAG architecture, often referred to as Multi-Step or Recursive RAG. This feature effectively functions as an autonomous research agent, capable of planning and executing a research strategy.

- **Workflow:**
    1. **Query Decomposition (Planning):** The user submits a broad research topic (e.g., "Trace the development of Logos theology from Greek philosophy through the Gospel of John and into early Trinitarian debates"). This complex query is first sent to a Gemini model with a specific instruction: "Break this research topic down into a logical sequence of sub-questions." The model might generate a plan like: 1. "What was the concept of Logos in Stoic and Platonic philosophy?" 2. "How is the term 'Logos' used in the prologue of the Gospel of John?" 3. "How did early church fathers like Irenaeus and Origen connect the Johannine Logos to the person of Christ?" 4. "What role did the Logos concept play in the formulation of the Nicene Creed?".
    2. **Iterative Retrieval (Execution):** The system then enters a loop, executing a RAG query for each sub-question. This is a form of stepwise reasoning where the output of one step can inform the next. For example, after retrieving information on Stoic philosophy, the system might refine the next query to specifically look for parallels between Stoic and Johannine language.
    3. **Contextual Refinement:** As each retrieval step completes, the retrieved chunks (text and metadata) are added to a growing "contextual workspace." The system continuously analyzes this workspace to identify what information has been gathered and what gaps remain, ensuring subsequent retrieval steps are focused and non-redundant.
    4. **Knowledge Graph Traversal:** For sub-questions involving influence or commentary (e.g., "How did Irenaeus interpret John's use of Logos?"), the system will leverage the relational metadata defined in Section 4. It can retrieve a passage from the Gospel of John and then perform a second-hop retrieval to find all texts in the corpus that have a relation_commentary or relation_intertextuality link pointing to that passage. This multi-hop capability is essential for tracing the flow of ideas.

5. **Final Synthesis (Writing):** Once the iterative retrieval process concludes (determined by completing the plan or reaching a predefined depth), the entire, rich contextual workspace is passed to the final Gemini generation model. The prompt for this final step is highly sophisticated, instructing the model to act as a scholar and synthesize all the gathered materials into a single, coherent, and well-structured research paper on the user's original topic. This process is asynchronous, and the user would be notified when their research report is ready.

This dual-architecture approach is a strategic necessity. It allows SCRIBE to offer both a fast, responsive chat experience and a powerful, deep research capability, using the appropriate level of architectural complexity for each task.

5.3 Techniques for Synthesizing Coherent, Thematic Narratives

The final synthesis step of the "Deep Research" feature is critical for producing high-quality output. Simply passing a large collection of text snippets to an LLM is not enough. The prompt must guide the model to structure its output logically and thematically.

- **Chain-of-Thought (CoT) Prompting:** The final synthesis prompt will incorporate CoT principles. It will instruct the LLM to "think step by step" by first creating an outline for its response, then identifying the major themes and arguments present in the provided context, and only then writing the full, flowing narrative. This structured reasoning process significantly improves the coherence and logical integrity of the final essay.
- **Persona-Based Generation:** The prompt will assign a specific, expert persona to the LLM, such as "You are a theological scholar and historian writing an entry for an academic encyclopedia." This instruction powerfully shapes the tone, vocabulary, style, and structural conventions of the generated text, ensuring it meets the user's expectations for a serious research document.

5.4 Citation and Source-Tracking for Verifiability

Trust and academic rigor are non-negotiable for Project SCRIBE. Every piece of information generated by the system must be verifiably grounded in the source corpus.

- **Systematic Tracking:** Throughout both the simple and multi-step RAG processes, the system will meticulously track the source of every chunk of information that is used. The id and canonical_reference from each chunk's metadata will be logged.
- **Instruction for Citation:** The final generation prompt will contain an explicit, non-negotiable instruction for the LLM to include inline citations for every claim it makes, using the canonical_reference provided with each context chunk.
- **Bibliography Generation:** At the end of every generated response (both chat answers and research papers), the system will automatically compile a bibliography or list of references. This list will be generated from the metadata of all the source chunks used in the response and will include the full title, author, and a direct source_url link to the original document, allowing users to instantly verify the information and explore the source texts for themselves. This commitment to transparent sourcing is the primary defense against model hallucination and is essential for building user trust.

Section 6: Orchestrating the Interaction: Advanced

Prompt Engineering

If the architecture is the hardware of Project SCRIBE, then the prompts are its software. The art and science of prompt engineering is what will translate the system's technical capabilities into nuanced, intelligent, and context-aware interactions. A sophisticated approach to prompting is required, moving beyond static templates to a dynamic "prompt pipeline" that assembles the optimal instructions for each specific task. This pipeline will use prompts not only for final answer generation but also to enhance the retrieval process itself.

6.1 Designing System Prompts and Personas for SCRIBE

At the start of every user interaction, a master system prompt will be loaded to establish the foundational behavior, persona, and constraints of the AI. This ensures consistency and aligns the model's operation with the project's core principles.

- **Core Persona:** The system prompt will define the AI's identity. For example: *"You are SCRIBE, a scholarly research assistant specializing in the Bible, ancient philosophy, and historical-theological texts. Your purpose is to help users explore the deep meaning of these works by providing objective, historically-grounded information. You must be respectful of all traditions and avoid expressing personal beliefs or theological opinions. Your responses must be generated solely from the provided context documents."* This persona-setting is crucial for managing user expectations and guiding the model's tone and style.
- **Core Constraints:** The system prompt will also include strict operational rules, often called "guardrails." For example: *"Constraint: Never invent information or answer questions for which you have not been provided relevant context. If the provided context does not contain the answer, you must state that the information is not available in the source documents. Constraint: You must provide a citation for every factual claim, referencing the source documents provided in the context."* These constraints are the primary technical mechanism for mitigating hallucinations and ensuring factual grounding.

6.2 Crafting Task-Specific Prompts

A single prompt template is insufficient for SCRIBE's dual features. The backend will dynamically construct prompts based on the user's task, assembling them from a library of pre-defined components.

- **Conversational Prompt Template:** This prompt will be optimized for conciseness, clarity, and low latency. It will follow a direct instruction format: *"Using only the following context documents, provide a clear and direct answer to the user's question. Present the answer first, followed by the supporting citations. Context: [...]. Question: [...]. Answer:"*.
- **Deep Research Synthesis Prompt Template:** This prompt will be significantly more complex, designed to guide the LLM through a multi-stage synthesis process. It will be structured to elicit a comprehensive and well-organized essay: *"You are a theological scholar tasked with writing a comprehensive research paper on the following topic. Synthesize the provided collection of research materials to construct your paper. Your paper should introduce the topic, identify and analyze the key themes, compare and contrast different viewpoints found in the sources, and conclude with a summary of the concept's development. Use formal academic language and provide inline citations for all*

claims. Topic: [...]. Research Materials: [...]. Paper:".
This modular approach, where the system assembles prompts from components like a system prompt, a task-specific instruction, and formatted context, creates a flexible and powerful "prompt pipeline" that can adapt to any user request.

6.3 Advanced Prompting Techniques for Retrieval and Generation

To achieve state-of-the-art performance, SCRIBE will employ several advanced prompting techniques that are integrated throughout the RAG pipeline.

- **Chain-of-Thought (CoT):** For complex reasoning tasks, particularly within the Deep Research feature, the prompt will explicitly instruct the model to "think step by step." This can be done by prepending the phrase to the prompt or by asking the model to first output its reasoning process or an outline before generating the final answer. This technique has been shown to significantly improve the accuracy of LLMs on complex, multi-step problems.
- **Few-Shot Prompting:** When a specific output format is required (e.g., a table comparing the views of two philosophers), the system will use few-shot prompting. The prompt will include one or two complete examples of the desired input-output format, which effectively teaches the model what is expected in the current context. This is far more reliable than trying to describe the format in words alone.
- **Query Rewriting & Expansion (HyDE):** Advanced prompting is not limited to the generation step; it can also enhance retrieval. For certain queries, the system can employ a technique like Hypothetical Document Embeddings (HyDE). In this process, before performing the vector search, the system first sends the user's query to an LLM and asks it to generate a *hypothetical* document that would perfectly answer that query. This generated document, which is often more detailed and context-rich than the original query, is then embedded. The resulting vector is used for the similarity search. This often leads to more semantically relevant results because the hypothetical document bridges the gap between the user's concise query and the verbose text in the corpus. This blurs the line between retrieval and prompting, using an LLM to improve the input to the RAG process itself.

6.4 Managing Dialogue History for Contextual Follow-up

For the Conversational Chat to feel natural, it must understand conversational context and handle follow-up questions. The system will maintain a short-term memory of the recent turns in the conversation.
When a user asks a follow-up question (e.g., "What about his views on justice?"), the system will not use this ambiguous phrase for retrieval. Instead, it will perform a preliminary, lightweight LLM call. This call will combine the conversation history with the new user question and generate a new, self-contained query. For example, if the previous turn was about Plato's *Republic*, the system would generate a new query like: "What were Plato's views on justice as described in The Republic?" This rewritten, context-aware query is then sent through the full RAG pipeline, ensuring that the retrieved documents are relevant to the actual, implicit topic of the conversation.

Section 7: Project Phasing, Governance, and Cost

Optimization

A project of SCRIBE's ambition requires a pragmatic and disciplined approach to development, deployment, and maintenance. This involves a phased rollout to deliver value incrementally, a robust MLOps framework to ensure quality and stability, and a proactive cost management strategy to maintain financial viability. This section outlines the practical roadmap for bringing Project SCRIBE to life.

7.1 A Phased Rollout Plan

An iterative development methodology will be adopted. This approach allows for early feedback, reduces risk, and ensures that foundational components are solid before building more complex features on top of them.

Phase	High-Level Timeline	Key Activities	Key Deliverables	Primary Success Metric
1: Foundation & MVP	Q1-Q2	- Set up core GCP architecture (GCS, Vertex AI, Cloud Run). - Implement IaC with Terraform. - Build ingestion pipeline for a single source (e.g., KJV Bible). - Implement hierarchical chunking and basic metadata schema. - Develop a simple, single-step RAG loop.	- Internal-only chat prototype. - Indexed corpus of at least one major work. - Initial "golden dataset" for evaluation.	Successful ingestion and accurate retrieval for 90% of the golden dataset questions for the initial corpus.
2: Corpus Expansion & Beta	Q3-Q4	- Build ingestion pipelines for all major sources (Gutenberg, Archive.org, Sefaria). - Implement the full, rich metadata schema, including relational data. - Develop the public-facing	- Public beta of the Conversational Assistant. - Corpus expanded to >1,000 key texts. - User feedback collection mechanism.	Achieve 1,000+ daily active users (DAU) within the first month of beta. Positive user feedback score > 80%.

Phase	High-Level Timeline	Key Activities	Key Deliverables	Primary Success Metric
		Conversational Chat Assistant UI. - Implement dialogue history management for follow-up questions.		
3: Deep Research & Full Release	Q5	- Develop and test the multi-step/recursive RAG architecture. - Implement advanced retrieval techniques (e.g., query decomposition, HyDE). - Build the UI for submitting research topics and viewing results. - Launch the "Deep Research" generator feature.	- Full public release of Project SCRIBE with both core features. - Comprehensive documentation and user guides.	Deep Research feature successfully generates coherent, cited essays for 85% of complex test queries.
4: Curation & Optimization	Ongoing	- Develop and deploy the human-in-the-loop curation tools. - Begin systematic metadata enrichment by domain experts. - Experiment with fine-tuning embedding or generation models. - Continuously monitor and optimize	- A fully operational data curation workflow. - Regular updates to the corpus and models. - Published performance and cost optimization reports.	Measurable improvement in retrieval/generation metrics (e.g., 5% increase in faithfulness score). 10% reduction in cost-per-query.

Phase	High-Level Timeline	Key Activities	Key Deliverables	Primary Success Metric
		performance and cost.		
Table 7.1: Phased Rollout Plan for Project SCRIBE				

7.2 MLOps and CI/CD for a RAG-based System

To manage the complexity of a RAG system in production, a robust MLOps (Machine Learning Operations) framework is not a luxury but a necessity. It ensures automation, reproducibility, and quality control throughout the project lifecycle.

- **Infrastructure as Code (IaC):** All GCP resources will be defined and managed using Terraform. This allows the entire architecture to be version-controlled, easily replicated across different environments (dev, staging, prod), and recovered quickly in case of disaster.
- **CI/CD Pipelines:** Automated Continuous Integration/Continuous Deployment (CI/CD) pipelines will be established using Google Cloud Build. When code is committed to the repository, these pipelines will automatically build container images, run unit and integration tests, and deploy the updated services to Cloud Run.
- **Model & Data Versioning:** All assets will be strictly versioned. Container images will be stored in Artifact Registry. The "golden dataset" used for evaluation, along with major versions of the ingested corpus, will be versioned in Google Cloud Storage. This ensures that any experiment or production run can be precisely reproduced.
- **Automated Testing & Evaluation:** The CI/CD pipeline will include an automated evaluation step. Before deploying any change that could affect output quality (e.g., a new prompt template, a different chunking strategy), the system will be run against the "golden dataset." The new responses will be automatically compared against the benchmark responses, and a quality score will be generated. This prevents regressions and provides a quantitative measure of whether a change is an improvement. This "golden dataset" is a core strategic asset, transforming the subjective question of "is the model better?" into an objective, engineering-driven process.

7.3 Cost Management and Optimization Strategies for GCP AI/ML

Generative AI workloads can be resource-intensive, and proactive cost management is essential for the project's long-term sustainability. The most significant and often underestimated cost risk in a large-scale RAG system is not the inference calls, but the data ingestion and re-embedding process. An inefficient pipeline that repeatedly re-processes the entire corpus can lead to runaway costs. Therefore, cost optimization efforts will focus on both the ingestion and inference stages.

- **Monitoring & Budgeting:**
    - **Granular Cost Tracking:** All GCP resources will be assigned labels (e.g., component:ingestion, feature:chat, env:prod) to allow for precise cost attribution. Billing data will be exported to BigQuery for detailed, query-based analysis of spending patterns.
    - **Budgets and Alerts:** Strict budgets will be set in the Google Cloud Billing console

for different projects and services. Automated alerts will be configured to notify the team when spending approaches or exceeds predefined thresholds, preventing unexpected cost overruns.

- **Resource Optimization:**
  - **Leverage Serverless:** The heavy reliance on serverless services like Cloud Run and Cloud Functions is a key cost-control measure. These services can scale to zero, meaning the project incurs no compute costs when the system is idle.
  - **Committed Use Discounts (CUDs):** For any predictable, steady-state workloads (e.g., a minimum baseline of inference capacity), the project will purchase 1- or 3-year CUDs to achieve significant discounts over on-demand pricing.
  - **Data Storage Tiers:** Data will be stored in the most cost-effective GCS storage class based on its access frequency. For example, raw, unprocessed source files that are only accessed once during initial ingestion can be moved to Nearline or Coldline storage to reduce costs.
- **Pipeline and Application Optimization:**
  - **Efficient Ingestion:** As highlighted, this is paramount. The ingestion pipeline will be designed with selective re-indexing capabilities, ensuring that only new or changed documents are processed. A robust caching strategy for embeddings will also be implemented to avoid redundant API calls.
  - **Tiered LLM Usage:** The system will use different LLMs for different tasks based on a cost/performance trade-off. For internal, high-volume tasks like query rewriting or data summarization, a smaller, faster, and cheaper model (e.g., Gemini Flash) will be used. The most powerful and expensive models (e.g., Gemini 1.5 Pro) will be reserved for the final, user-facing generation step where quality is most critical.
  - **Batch Processing:** Ingestion of new documents will be done in batches wherever possible to reduce the per-job overhead of the processing pipeline.

7.4 Evaluation Framework: Measuring Success

A multi-faceted evaluation framework is required to measure the performance of a complex generative system like SCRIBE.

- **Retrieval Metrics:** The performance of the retrieval component will be evaluated against the golden dataset using standard information retrieval metrics, including Precision@k, Recall@k, and Mean Reciprocal Rank (MRR). These metrics quantify how well the system finds relevant documents.
- **Generation Metrics:** Evaluating the quality of the generated text is more complex. We will employ an "LLM-as-a-judge" methodology. A powerful, third-party LLM (like Gemini Pro or another state-of-the-art model) will be prompted to assess the generated responses based on several criteria:
  - **Faithfulness:** Does the answer accurately reflect the information in the provided source context?
  - **Relevance:** Does the answer directly address the user's query?
  - **Coherence:** Is the answer well-written, logical, and easy to understand?
- **Human Feedback:** Ultimately, the measure of success is user satisfaction. The UI will include a simple thumbs-up/thumbs-down mechanism and an optional comment field for each response. This direct feedback is invaluable for identifying systemic weaknesses, content gaps in the corpus, and areas for improvement in the user experience. This feedback loop is a critical component of the ongoing optimization phase.

Conclusion

The project plan outlined in this document provides a comprehensive blueprint for the development and deployment of Project SCRIBE. By adopting a hybrid "Full Control" architecture on Google Cloud, the project can achieve the deep customization required for its unique corpus while leveraging the scalability and reliability of managed services. The success of the project is predicated on a series of strategic decisions: the implementation of source-aware ingestion pipelines that preserve structured data; the use of a hierarchical, content-aware chunking strategy; the design of a rich metadata schema that actively participates in the reasoning process; and the development of distinct RAG architectures for its conversational and deep research features.

A disciplined, phased rollout, governed by robust MLOps practices and proactive cost management, will ensure the project is delivered in a sustainable and financially responsible manner. The commitment to verifiable, cited responses and a rigorous evaluation framework will build the user trust necessary for a tool of this nature.

Project SCRIBE has the potential to be more than just a question-answering system. By combining state-of-the-art Retrieval-Augmented Generation techniques with a deep and contextually rich corpus, it can become a powerful instrument for scholarly research and personal exploration, enabling users to engage with foundational texts of Western thought in a way that is both accessible and profound. This plan provides the technical and strategic foundation to realize that vision.

Works cited

1. RAG On GCP: Production-ready GenAI On Google Cloud Platform ..., https://xebia.com/blog/rag-on-gcp/ 2. RAG with Google Cloud Services : r/googlecloud - Reddit, https://www.reddit.com/r/googlecloud/comments/1cayxo3/rag_with_google_cloud_services/ 3. Vertex AI RAG Engine: A developers tool - Google Developers Blog, https://developers.googleblog.com/en/vertex-ai-rag-engine-a-developers-tool/ 4. Building Vertex AI RAG Engine with Gemini 2 Flash LLM | by Arjun Prabhulal | Google Cloud, https://medium.com/google-cloud/building-vertex-ai-rag-engine-with-gemini-2-flash-llm-79c27445dd48 5. Vertex AI RAG Engine overview - Google Cloud, https://cloud.google.com/vertex-ai/generative-ai/docs/rag-engine/rag-overview 6. Setup a RAG with Google Drive data using Google Cloud's RAG ..., https://medium.com/google-cloud/setup-a-rag-with-google-drive-data-using-google-clouds-rag-engine-84f932f315e8 7. What is Retrieval-Augmented Generation (RAG)? - Google Cloud, https://cloud.google.com/use-cases/retrieval-augmented-generation 8. Embeddings | Gemini API | Google AI for Developers, https://ai.google.dev/gemini-api/docs/embeddings 9. Using the Gemini embedding model to Develop a RAG System with observability via W&B Weave | genai-research – Weights & Biases - Wandb, https://wandb.ai/onlineinference/genai-research/reports/Using-the-Gemini-embedding-model-to-Develop-a-RAG-System-with-observability-via-W-B-Weave--VmlldzoxMzcyNzEyMg 10. Vertex AI APIs for building search and RAG experiences | AI Applications - Google Cloud, https://cloud.google.com/generative-ai-app-builder/docs/builder-apis 11. Cost Management | Google Cloud, https://cloud.google.com/cost-management 12. AI and ML perspective: Cost optimization | Cloud Architecture Center ...,

https://cloud.google.com/architecture/framework/perspectives/ai-ml/cost-optimization 13. Building an LLM and RAG-based chat application using AlloyDB AI and LangChain, https://codelabs.developers.google.com/codelabs/genai-db-retrieval-app 14. RAG in the Cloud: Comparing AWS, Azure, and GCP for Deploying Retrieval Augmented Generation Solutions, https://ragaboutit.com/rag-in-the-cloud-comparing-aws-azure-and-gcp-for-deploying-retrieval-augmented-generation-solutions/ 15. Is there a way to batch download the top 100 Project Gutenberg texts? For a gift for sister's graduation. : r/ebooks - Reddit, https://www.reddit.com/r/ebooks/comments/umypf/is_there_a_way_to_batch_download_the_top_100/ 16. Bulk downloader for free ebooks hosted at Project Gutenberg - GitHub, https://github.com/puntonim/gutenberg-bulk-downloader 17. My struggle to download every Project Gutenberg book in English : r/DataHoarder - Reddit, https://www.reddit.com/r/DataHoarder/comments/1ia3klh/my_struggle_to_download_every_project_gutenberg/ 18. Offline Catalogs | Project Gutenberg, https://www.gutenberg.org/ebooks/offline_catalogs.html 19. garethbjohnson/gutendex: Web API for Project Gutenberg ebook metadata - GitHub, https://github.com/garethbjohnson/gutendex 20. Help on Download Options and Bibliographic Record - Project Gutenberg, https://www.gutenberg.org/help/bibliographic_record.html 21. Internet Archive API Client • internetarchive - Docs - rOpenSci, https://docs.ropensci.org/internetarchive/articles/internet-archive.html 22. Access Archive-It's Wayback index with the CDX/C API, https://support.archive-it.org/hc/en-us/articles/115001790023-Access-Archive-It-s-Wayback-index-with-the-CDX-C-API 23. API Documentation · Sefaria/Sefaria-Project Wiki · GitHub, https://github.com/Sefaria/Sefaria-Project/wiki/API-Documentation/948bb3dcf283653163a2d0a6b88dca152cabaf76 24. Getting Started With The Sefaria API, https://developers.sefaria.org/reference/getting-started 25. Decoding Ancient History With AI - CHM, https://computerhistory.org/blog/decoding-ancient-history-with-ai/ 26. AI helps us decipher ancient texts, and in the process rewriting history | The Jerusalem Post, https://www.jpost.com/archaeology/archaeology-around-the-world/article-835867 27. Researchers Tap AI to Dig Into the Past – Communications of the ACM, https://cacm.acm.org/news/researchers-tap-ai-to-dig-into-the-past/ 28. Build Advanced Retrieval-Augmented Generation Systems ..., https://learn.microsoft.com/en-us/azure/developer/ai/advanced-retrieval-augmented-generation 29. Chunking Strategies for LLM Applications | Pinecone, https://www.pinecone.io/learn/chunking-strategies/ 30. Advanced RAG Techniques: an Illustrated Overview | by IVAN ILIN ..., https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6 31. ARAGOG: Advanced RAG Output Grading - arXiv, https://arxiv.org/html/2404.01037v1 32. Gemini Embedding: Powering RAG and context engineering - Google Developers Blog, https://developers.googleblog.com/en/gemini-embedding-powering-rag-context-engineering/ 33. Metadata Principles and Practicalities - D-Lib Magazine, https://www.dlib.org/dlib/april02/weibel/04weibel.html 34. DESCRIPTIVE METADATA FOR ELECTRONIC RECORDS - the State Historical Society of North Dakota, https://www.history.nd.gov/archives/MetadataTutorial.pdf 35. Advanced RAG and the 3 types of Recursive Retrieval | by Chia Jeng Yang - Medium, https://medium.com/enterprise-rag/advanced-rag-and-the-3-types-of-recursive-retrieval-cdd0fa52e1ba 36. What is Multi-Step RAG (A Complete Guide) - F22Labs, https://www.f22labs.com/blogs/what-is-multi-step-rag-a-complete-guide/ 37. A Complete Guide to Implementing Recursive/Multi-Step RAG | by Gaurav Nigam - Medium,

https://medium.com/aingineer/a-complete-guide-to-implementing-recursive-multi-step-rag-5afca90f57ee 38. Beyond Basic Prompts: Exploring the Nuances of Prompt Engineering in Artificial Intelligence | by Amina Javaid | Medium, https://medium.com/@aminajavaid30/beyond-basic-prompts-exploring-the-nuances-of-prompt-engineering-in-artificial-intelligence-0be2adfcb9b5 39. What is Prompt Engineering? A Detailed Guide For 2025 - DataCamp, https://www.datacamp.com/blog/what-is-prompt-engineering-the-future-of-ai-communication 40. Effective Prompts for AI: The Essentials - MIT Sloan Teaching ..., https://mitsloanedtech.mit.edu/ai/basics/effective-prompts/ 41. Prompt Engineering for AI Guide | Google Cloud, https://cloud.google.com/discover/what-is-prompt-engineering 42. What Is Prompt Engineering? | IBM, https://www.ibm.com/think/topics/prompt-engineering 43. Crafting Conversations with AI: The Magic of Prompt Engineering | by Akim | Medium, https://medium.com/@akim48178/crafting-conversations-with-ai-the-magic-of-prompt-engineering-1217c21beb03 44. Advanced RAG techniques part 2: Querying and testing - Elasticsearch Labs, https://www.elastic.co/search-labs/blog/advanced-rag-techniques-part-2 45. Managing Costs on Google Cloud Platform: Tips for Saving Money - HAKIA.com, https://www.hakia.com/posts/managing-costs-on-google-cloud-platform-tips-for-saving-money 46. Generative AI Infrastructure Costs: A Practical Guide to GCP, Azure, AWS, and Beyond | by Jitendra Gupta | Cloud Experts Hub | Medium, https://medium.com/cloud-experts-hub/generative-ai-infrastructure-costs-a-practical-guide-to-gcp-azure-aws-and-beyond-fafb2808b1af 47. GCP Cost Optimization: 7 Essential Tactics for 2025 - Cast AI, https://cast.ai/blog/gcp-cost-optimization/