Comprehensive Architectural Specification: 'Autobiography App: From Speech to AI'

1. Executive Summary

The preservation of personal history has traditionally been a bifurcated endeavor: either a labor-intensive literary exercise requiring significant writing skill or an ephemeral oral tradition lost to time. The 'Autobiography App: From Speech to AI' project represents a paradigm shift in this domain, proposing a system that leverages the multimodal capabilities of modern Artificial Intelligence to democratize legacy creation. By utilizing the Google Cloud Platform (GCP) and the Gemini family of models, this project aims to create a "Google-Native" ecosystem where the human voice is the primary interface for generating structured, searchable, and narrative-rich autobiographies.

This report details the exhaustive methodology for engineering this system. Unlike traditional dictation software, which merely converts speech to text, this architecture is designed to function as an active "Biographer Agent." It captures raw audio, extracts semantic meaning, identifies temporal and entity relationships, and synthesizes disjointed memories into a cohesive life story. The technical backbone relies strictly on Google Cloud services, utilizing **Flutter** for cross-platform mobile delivery, **Firebase** for serverless backend orchestration, **Cloud Firestore** for NoSQL data modeling, and **Vertex AI** (specifically Gemini 1.5 Pro and Flash) for cognitive processing.

The implementation roadmap is segmented into three distinct maturity phases. Phase 1 (MVP) focuses on the development of a "Capture Engine" capable of ingesting long-form audio and performing high-fidelity transcription with speaker diarization. Phase 1.5 introduces the "Structuring Engine," utilizing Gemini's entity extraction capabilities to build a queryable "Life Graph" of people, places, and timelines. Finally, Phase 2 realizes the "Narrative Engine," implementing Retrieval-Augmented Generation (RAG) and vector similarity search to enable an autonomous AI interviewer that detects gaps in the user's story and generates synthesized book chapters.

Critical analysis of the research indicates that the primary technical challenge lies not in transcription, but in *contextual continuity*. Standard Speech-to-Text (STT) APIs lack the long-context understanding required to link a childhood memory mentioned in Session 1 with a career choice discussed in Session 10. The proposed solution addresses this by bypassing traditional STT in favor of Gemini 1.5 Pro's multimodal ingestion, which supports context windows of up to 2 million tokens, effectively allowing the AI to "remember" the entire scope of the user's life during processing. This report provides the blueprint for this architecture, defining the specific data models, prompt engineering strategies, and infrastructure configurations required to execute this vision.

2. Methodology: The Intersection of Oral History and Generative AI

To architect a successful autobiography application, one must first understand the theoretical frameworks of oral history and how they map to Generative AI capabilities. The system cannot

simply be a passive recorder; it must embody the techniques of a professional oral historian.

2.1 The Digital Oral History Framework

Research into best practices for oral history reveals that a successful interview is not a random conversation but a structured traversal of memory. Professional historians utilize a "Question Tree" structure, moving from foundational questions ("Where were you born?") to thematic explorations ("How did the war affect your family?"). The methodology for this application translates these human-centric practices into algorithmic logic.
The application adopts a **Thematic-Chronological Hybrid Model**. While life occurs chronologically, memory is often associative. A user discussing their wedding (Chronological: Adulthood) may suddenly recall their parents' marriage advice (Chronological: Childhood). The system must support this non-linear flow while maintaining a linear database structure.

- **The Pre-Interview Context**: Just as a historian conducts background research , the AI utilizes a "Profile Ingestion" phase where users upload basic metadata (DOB, Hometown). This primes the model's system instructions to avoid generic questions.
- **The Socratic Interview Method**: The AI is instructed to use open-ended prompts ("Tell me about...") rather than binary questions. When the user responds, the system employs **active listening algorithms**—implemented via RAG—to ask follow-up questions derived specifically from the immediate content, rather than following a rigid script.

2.2 Algorithmic Memory Traversal

The core methodological innovation is the digitization of the "Interview Question Tree." We treat the user's life as a graph where nodes represent life stages (Early Childhood, Education, Career, Family, Twilight Years) and edges represent transitional events.

- **Tree Traversal Algorithms**: The system utilizes a modified Breadth-First Search (BFS) to ensure broad coverage of all life stages in the early phases, switching to Depth-First Search (DFS) when a user shows high engagement in a specific topic (e.g., spending 20 minutes discussing a specific job).
- **Missing Node Identification**: A critical function of the historian is identifying gaps. "You mentioned your mother, but never your father." The methodology employs a "Coverage Analysis" pass where Gemini evaluates the accumulated transcripts against a standard "Life Schema." Missing nodes (e.g., "No mention of siblings") trigger the generation of specific prompts in future sessions.

3. Technical Architecture: The Google Cloud Ecosystem

The architectural mandate is a strict adherence to the Google Cloud Platform (GCP) stack. This decision is strategic, leveraging the deep integration between Firebase (mobile development) and Vertex AI (enterprise intelligence) to reduce latency and development overhead.

3.1 The Client Layer: Flutter and Firebase

The frontend is constructed using **Flutter**, Google's UI toolkit. Flutter is uniquely suited for this application due to its high-performance rendering engine, which is necessary for complex timeline visualizations, and its unified Dart codebase which simplifies data object sharing with the backend.

- **Firebase Integration**: The application utilizes the flutterfire suite of packages.
    - **Authentication**: Firebase Auth handles user identity, supporting anonymous sign-ins (critical for reducing friction during initial "try-out" sessions) that can be later upgraded to permanent accounts linked to Google or Apple IDs.
    - **State Management**: Given the complex state of long-running audio recordings and real-time AI responses, the **Bloc (Business Logic Component)** pattern is recommended. This separates the presentation layer (UI) from the business logic (Audio Recorder, AI Service), ensuring that a UI re-render does not interrupt an active recording session.

3.2 The Backend Layer: Serverless Orchestration

The backend architecture is entirely serverless, relying on **Firebase Data Connect** and **Cloud Functions (2nd Gen)**. This eliminates the need for server provisioning and scaling, allowing the infrastructure to scale down to zero cost when inactive—a crucial economic factor for a consumer app.

- **Cloud Functions (Gen 2)**: These are the workhorses of the application. Unlike Gen 1 functions, Gen 2 functions are built on Cloud Run, offering significantly longer timeout durations (up to 60 minutes). This is a non-negotiable requirement for an app processing audio files that may be 30-60 minutes in length. A standard 9-minute HTTP timeout would cause failures during the transcription of long oral history sessions.
- **Firebase Data Connect**: This newly introduced service bridges Firebase with Cloud SQL (PostgreSQL), but for this architecture, we lean on **Cloud Firestore** for its document-oriented nature which maps 1:1 with JSON-based AI outputs. However, Data Connect's ability to provide type-safe SDKs is emulated through strict Firestore data converters in the Flutter client.

3.3 The Intelligence Layer: Vertex AI and Gemini

The core differentiation of this project is the rejection of standard "API chaining" (Speech-to-Text -> NLP -> Summary) in favor of a single **Multimodal Model** approach using Gemini.

- **Gemini 1.5 Pro**: This model serves as the "Heavy Lifter." Its 1M+ token context window allows it to ingest raw audio files directly. This enables "Native Audio Understanding," where the model perceives tone, cadence, and emotion—nuances lost in standard text transcriptions.
- **Vertex AI for Firebase SDK**: This SDK allows the mobile client to communicate directly with Gemini for low-latency tasks (like generating a quick title for a recording or a chat response) while maintaining security via Firebase App Check. This removes the latency of hopping through a custom backend API for real-time interactions.

3.4 Data Storage Strategy

- **Cloud Storage for Firebase**: This stores the "Golden Copy" of the raw audio. We utilize the audio/x-m4a MIME type for efficiency. Lifecycle policies are configured to transition files to "Nearline" storage after 30 days to optimize costs, as the text transcript becomes the primary active data source.
- **Vector Storage**: Embedding vectors (generated via the Vertex AI text-embeddings API) are stored directly within Firestore documents, enabling K-Nearest Neighbor (KNN) searches without managing a separate vector database like Milvus or Pinecone.

4. Phase 1 Implementation: The Capture Engine (MVP)

The Minimum Viable Product (MVP) is defined by its ability to reliably capture the user's voice and convert it into a highly accurate, diarized transcript. This phase establishes the "Data Ingestion" pipeline.

4.1 Audio Acquisition in Flutter

The mobile implementation prioritizes audio fidelity and recording stability. We utilize the flutter_sound or record package to interface with the device microphone.
Audio Configuration:

- **Format**: AAC (Advanced Audio Coding) wrapped in an M4A container.
- **Bitrate**: 64kbps Mono. This is sufficient for voice clarity while keeping file sizes manageable (approx. 28MB per hour) for mobile uploads.
- **Sample Rate**: 16kHz or 44.1kHz. While 16kHz is sufficient for ASR (Automated Speech Recognition), 44.1kHz is preferred if the user intends to export the audio as a podcast later.

Reliability Engineering:

- **Local Buffering**: Audio is written directly to the local file system (Temporary Directory) during recording. This protects against data loss if the app crashes.
- **Foreground Service**: On Android, a foreground service notification is mandatory to prevent the OS from killing the recording process when the screen turns off.
- **Resumable Uploads**: Using firebase_storage, we implement resumable uploads. If the user's network drops, the upload pauses and resumes automatically, a critical feature for long files.

4.2 The Gemini Ingestion Pipeline

Upon the successful upload of an audio file to Cloud Storage, a **Cloud Function** is triggered via the google.storage.object.finalize event. This function orchestrates the AI processing.
Architectural Pivot: Multimodal vs. ASR Traditional architectures would send this file to the Google Cloud Speech-to-Text API. However, research indicates that Gemini 1.5 Pro offers superior performance for "understanding" audio. Cloud Speech-to-Text provides a stream of words; Gemini provides a structured analysis.

- **Cost/Benefit**: While Gemini token costs can be higher for pure transcription, it collapses three distinct operations (Transcription, Speaker Diarization, Sentiment Analysis) into a single API call, reducing overall system complexity and latency.

Implementation of the Cloud Function: The function uses the **Vertex AI Node.js SDK**. It constructs a request that includes the Cloud Storage URI (gs://...) of the audio file.

- **Prompt Engineering**: The prompt is designed to force a structured JSON output. This is crucial for programmatic parsing.

```
const prompt = `
You are an expert transcriber. Process the audio file and generate
a JSON response.
Schema:
{
  "transcript": "Full verbatim text...",
  "summary": "Brief 3-sentence summary...",
  "speakers":,
  "segments":
```

```
}
Requirements:
1. Identify distinct speakers.
2. Detect emotion in speech.
3. Do not summarize the transcript; provide it verbatim.
`;
```

This single-shot prompt achieves what would otherwise require a complex pipeline of separate ML models.

4.3 Handling Long Audio Contexts

Gemini 1.5 Pro supports up to 9.5 hours of audio in a single prompt. This eliminates the need for complex "chunking" logic (splitting audio into 5-minute segments) that is often required with other LLMs. However, for extreme cases or to optimize costs, we can implement a logic check in the Cloud Function:

- If file_size < 50MB: Send to Gemini Pro directly.
- If file_size > 50MB: Use ffmpeg (via a localized binary in the Cloud Function environment) to downsample the bitrate before sending, ensuring we fit within the model's bandwidth constraints without splitting the narrative context.

5. Phase 1.5 Implementation: The Structuring Engine

Once the text is captured, the system must organize it. Phase 1.5 transforms a linear list of recordings into a multi-dimensional "Life Graph."

5.1 Data Modeling: The Memory Node

In Firestore, we define the atomic unit of the application as a **Memory Node**. **Schema Definition**:

- collection: users/{userId}/memories
- documentId: Auto-generated UUID.
- fields:
  - transcript (String): The full text.
  - audioRef (String): GS URI.
  - processedAt (Timestamp).
  - temporalMetadata (Map): Stores the "When".
  - spatialMetadata (Map): Stores the "Where".
  - entities (Array<String>): IDs of people/topics mentioned.

5.2 Temporal Logic and Fuzzy Dates

Autobiographies rarely contain precise dates. Users say "Summer of '69" or "When I was ten," not "July 12, 1969." The system requires a logic layer to convert these semantic dates into sortable timestamps.

- **Inference Algorithm**:
  1. Retrieve User Profile (e.g., DOB: 1950-01-01).
  2. Gemini extracts the temporal phrase: "When I was ten."
  3. Cloud Function calculates: 1950 + 10 = 1960.
  4. Store: estimatedDate: 1960-06-01 (Mid-year default), precision: "year".
- **Indexing**: We create a composite index in Firestore on userId and estimatedDate. This

allows the Flutter client to query memories.where('userId', '==', uid).orderBy('estimatedDate') to render the timeline.

5.3 Entity Extraction and Management

To enable features like "Show me all stories about my Mother," we use Gemini's Named Entity Recognition (NER) capabilities.

- **Extraction Prompt**: "Identify all unique people, places, and organizations in this text. Return them as a JSON list."
- **Resolution Logic**: When Gemini returns "Mom," the backend must resolve this to a unique Entity ID.
    - Query users/{uid}/entities where aliases contains "Mom".
    - If found: Link Memory to existing Entity ID.
    - If not found: Create new Entity "Mom" and add to the graph. This creates a relational web within the NoSQL structure, essential for the "Thematic View" in the UI.

5.4 Flutter UI: The Vertical Timeline

Visualization is key to the user experience. We implement a vertical timeline using the timelines_plus package or a custom CustomPaint widget.

- **Gap Visualization**: The UI logic iterates through the sorted list of memories. If the gap between Memory[n].endDate and Memory[n+1].startDate > 5 years, the app inserts a "Missing Era" widget.
- **Interaction**: Tapping this "Missing Era" widget triggers the AI Interviewer: "You have a gap between 1980 and 1985. Would you like to talk about that period?"

6. Phase 2 Implementation: The Narrative Engine (RAG)

Phase 2 represents the transition from "Smart Recorder" to "AI Biographer." This requires the system to "remember" previous conversations and actively guide the user to complete their story.

6.1 Vector Embeddings and Semantic Search

To implement memory, we utilize **Vector Embeddings**. An embedding is a numerical representation (a list of floating-point numbers) of the semantic meaning of text.

- **Generation**: We use the **Vertex AI Text-Embeddings API** (Gecko model). A Cloud Function triggers whenever a Memory Node is created or updated. It sends the summary text to the Gecko model and receives a 768-dimensional vector.
- **Storage**: This vector is stored in the embedding field of the Firestore document.
- **Indexing**: We enable **Firestore Vector Search** by creating a specialized vector index.

```
gcloud firestore indexes composite create \
--collection-group=memories \
--query-scope=COLLECTION \
--field-config field-path=userId,order=ASCENDING \
--field-config field-path=embedding,vector-
config='{"dimension":"768", "flat": "{}"}'
```

This configuration is critical. It allows us to perform a Nearest Neighbor search *scoped* to a specific user. Without the userId field config, the search would scan the entire database (all users), violating privacy and destroying performance.

6.2 Retrieval-Augmented Generation (RAG) Workflow

The RAG workflow enables the "Interviewer Mode."

1. **User Context**: The user opens the app and says, "I want to talk about my career."
2. **Query Vectorization**: The app converts the query "career" (and recent conversation history) into a vector.
3. **Retrieval**: Firestore performs a KNN search to find the 5 most relevant past memories.
    - *Result*: A memory about "First job at 16" and "College graduation."
4. **Prompt Augmentation**: The system constructs a prompt for Gemini:
    - *System*: "You are a biographer."
    - *Context*: "The user has previously discussed their first job (Memory A) and college (Memory B)."
    - *Task*: "Ask a question that bridges the gap between College and their first professional role."
5. **Output**: "You mentioned working at the grocery store at 16, and then graduating college in 1970. What was your first 'real' job after graduation?" This creates the illusion of a continuous, attentive listener.

6.3 Dynamic Interview Logic

We implement a "Generative State Machine" for the interview process.

- **The Tree**: A JSON structure represents the ideal autobiography (Nodes: Roots, Childhood, Adolescence, Early Career, etc.).
- **Progress Tracking**: A progress map in the User profile tracks the semantic coverage of each node.
- **Tree Traversal**: The AI selects the next "Unvisited" node in the tree.
- **Question Generation**: Using the RAG context, it generates a question specific to that node.
    - *Standard*: "Tell me about your siblings."
    - *RAG-Enhanced*: "You mentioned your brother John in the story about the farm. Did you have other siblings?".

6.4 Narrative Synthesis: Chapter Generation

The final output is a book. This requires transforming spoken transcripts into written prose.

- **Map-Reduce Strategy**:
    - **Map**: Process each Memory Node individually. "Convert this spoken transcript into a first-person narrative, removing filler words and improving grammar, but retaining the user's voice."
    - **Reduce**: Combine chronologically adjacent nodes into Chapters.
- **Contextual Smoothing**: A final pass with Gemini 1.5 Pro (using its large context window) reads the generated chapter to smooth transitions. It identifies disjointed paragraphs and inserts transitional sentences ("Years later, this event would make sense, but at the time...").
- **Style Transfer**: The user can select a "Voice" (e.g., "Hemingway-esque", "Conversational", "Formal"). This is passed as a style guide in the System Instruction.

7. Operational Strategy: Security, Privacy, and Costs

7.1 Security Architecture

- **Firebase App Check**: Mandatory implementation to prevent abuse of the AI APIs. It ensures that API calls originate from the genuine iOS/Android app.
- **Row-Level Security**: Firestore Security Rules strictly enforce that users can only read/write their own memories.

```
match /users/{userId}/memories/{document=**} {
  allow read, write: if request.auth.uid == userId;
}
```

- **Data Encryption**: All data is encrypted at rest in Firestore and Storage. Audio files use signed URLs with short expiration times for playback.

7.2 Cost Modeling and Optimization

AI services can be expensive. A strategic approach is required to maintain unit economics.

- **Gemini Flash vs. Pro**:
  - Use **Gemini 1.5 Pro** only for the initial Transcription (where audio understanding is key) and Final Book Generation (where quality matters).
  - Use **Gemini 1.5 Flash** for all intermediate steps: Title generation, Entity Extraction, Chat interaction. Flash is significantly cheaper and faster.
- **Vector Optimization**: Do not vectorize the entire transcript. Vectorize the *Summary*. This reduces the token count sent to the Embedding API and creates denser, more relevant vectors for search.
- **Storage Lifecycle**: Audio files are the largest cost driver. Implement a lifecycle rule to move audio to "Archive" storage (Coldline) 30 days after transcription. The text transcript is the active data; the audio is merely a backup artifact.

7.3 Privacy Compliance

- **Training Data**: It is imperative to explicitly configure Vertex AI not to use customer data for model training. Google Cloud provides this guarantee for enterprise Vertex AI users, distinguishing it from consumer-grade tools.
- **Right to be Forgotten**: A "Delete Account" function must trigger a recursive deletion Cloud Function that purges the User document, all Memories, all Entities, and the associated Storage bucket folder.

8. Conclusion

The 'Autobiography App: From Speech to AI' represents a sophisticated convergence of cloud infrastructure and generative intelligence. By strictly adhering to the Google Cloud technology stack, the project benefits from the seamless interoperability of Firebase and Vertex AI, allowing for a development velocity that would be impossible with a fragmented vendor approach.
The methodology outlined proceeds logically from high-fidelity capture (MVP) to semantic structuring (V1.5) and finally to autonomous narrative generation (V2). Key architectural decisions—such as the use of Gemini 1.5 Pro for direct audio ingestion and the implementation of user-scoped Vector Search in Firestore—solve the fundamental challenges of context and

continuity in oral history.
This report serves as a definitive technical specification. It moves beyond abstract concepts to define the specific schemas, indexes, and prompt strategies required to build a system that does not merely record a voice, but preserves a life.

9. Appendix: Data & Implementation Reference

Table 1: Firestore Schema Specification

Collection Path	Document ID	Key Fields	Description
users	{uid}	dob, hometown, writingStyle, treeProgress	User profile and global settings.
users/{uid}/memories	{memoryId}	transcript, summary, audioUri, embedding, estimatedDate	The core atomic unit of the story.
users/{uid}/entities	{entityId}	name, type, mentions (Array of Refs)	Graph nodes for People, Places, Topics.
users/{uid}/chapters	{chapterId}	title, content, dateRange, memoryRefs	Synthesized narrative text (V2).

Table 2: Google Cloud Service Utilization Matrix

Component	GCP Service	Justification
App Framework	Flutter	Single codebase, tight Firebase integration.
Auth	Firebase Auth	Anonymous -> Registered conversion flow.
Database	Cloud Firestore	Flexible schema, native Vector Search support.
Object Storage	Cloud Storage	Resumable uploads for large audio files.
Compute	Cloud Functions (Gen 2)	Long timeouts (60m) for AI processing.
AI (Cognitive)	Vertex AI (Gemini 1.5 Pro)	Multimodal audio ingestion, massive context window.
AI (Embeddings)	Vertex AI (Gecko)	Semantic vector generation for RAG.
AI (Interactive)	Firebase Vertex SDK	Client-side secure AI calls for chat.

Code Snippet: Firestore Composite Index Configuration (Terraform)

```
resource "google_firestore_index" "user_memory_vector_index" {
  project = "autobiography-project-id"
  database = "(default)"
  collection = "memories"
```

```
  fields {
    field_path = "userId"
    order      = "ASCENDING"
  }

  fields {
    field_path = "embedding"
    vector_config {
      dimension = 768
      flat {}
    }
  }
}
```

Note: This specific configuration enables the critical privacy feature of searching vectors ONLY within a specific user's dataset.

Works cited

1. Oral History Best Practices, https://oralhistory.org/best-practices/ 2. CONDUCTING AN ORAL HISTORY INTERVIEW - Sustainable Heritage Network, https://sustainableheritagenetwork.org/system/files/atoms/file/Oral_History_Planning_and_Setup_%20Workflow.pdf 3. Easy and Fun Autobiography Outline Template - Meminto Stories, https://meminto.com/blog/easy-and-fun-autobiography-outline-template/ 4. Vertex AI in Firebase Aims to Simplify the Creation of Gemini-powered Mobile Apps - InfoQ, https://www.infoq.com/news/2024/10/vertex-ai-firebase-gemini/ 5. Tree cheatsheet for coding interviews - Tech Interview Handbook, https://www.techinterviewhandbook.org/algorithms/tree/ 6. The ultimate guide to tree traversals for interviews | by The Educative Team, https://learningdaily.dev/the-ultimate-guide-to-tree-traversals-for-interviews-7260e517f56f 7. Towards Detecting Prompt Knowledge Gaps for Improved LLM-guided Issue Resolution, https://arxiv.org/html/2501.11709v1 8. Firebase Studio lets you build full-stack AI apps with Gemini | Google Cloud Blog, https://cloud.google.com/blog/products/application-development/firebase-studio-lets-you-build-full-stack-ai-apps-with-gemini 9. Audio understanding | Gemini API - Google AI for Developers, https://ai.google.dev/gemini-api/docs/audio 10. Multimodal Prompting with Gemini: Working with Audio - Google Cloud Applied AI Engineering, https://googlecloudplatform.github.io/applied-ai-engineering-samples/genai-on-vertex-ai/gemini/prompting_recipes/multimodal/multimodal_prompting_audio/ 11. Firebase AI Logic client SDKs | Build generative AI features directly into your mobile and web apps, https://firebase.google.com/products/firebase-ai-logic 12. Building AI-powered apps with Firebase AI Logic, https://firebase.blog/posts/2025/05/building-ai-apps/ 13. Upload files with Cloud Storage on Flutter - Firebase, https://firebase.google.com/docs/storage/flutter/upload-files 14. How to record voice and store them to firebase storage in flutter? - Stack Overflow, https://stackoverflow.com/questions/64849439/how-to-record-voice-and-store-them-to-firebase-storage-in-flutter 15. The Best Speech Recognition API in 2025: A Head-to-Head Comparison | Voice Writer Blog, https://voicewriter.io/blog/best-speech-recognition-api-2025 16. Audio understanding (speech only) | Generative AI on Vertex AI | Google Cloud Documentation, https://docs.cloud.google.com/vertex-ai/generative-ai/docs/multimodal/audio-understanding 17. Index types in Cloud Firestore - Firebase - Google, https://firebase.google.com/docs/firestore/query-data/index-overview 18. Search with vector embeddings | Firestore - Firebase - Google, https://firebase.google.com/docs/firestore/vector-search 19. Structured Output with Gemini Models: Begging, Threatening, and JSON-ing | by

Saverio Terracciano | Google Cloud - Medium, https://medium.com/google-cloud/structured-output-with-gemini-models-begging-borrowing-and-json-ing-f70ffd60eae6 20. Building Stunning Timelines in Flutter: A Developer's Guide | by Hassan | Medium, https://medium.com/@hassanisrar15/building-stunning-timelines-in-flutter-a-developers-guide-1ec519bd9eab 21. timelines_plus | Flutter package - Pub.dev, https://pub.dev/packages/timelines_plus 22. Perform vector similarity search with Vertex AI | Data Connect - Firebase - Google, https://firebase.google.com/docs/data-connect/solutions-vector-similarity-search 23. Get started with Firestore vector similarity search | Google Cloud Blog, https://cloud.google.com/blog/products/databases/get-started-with-firestore-vector-similarity-search 24. Retrieval-augmented generation (RAG) | Genkit - Firebase - Google, https://firebase.google.com/docs/genkit/rag 25. Building a RAG application with vector search in Firestore | by Suds Kumar | Google Cloud, https://medium.com/google-cloud/building-a-rag-application-with-vector-search-in-firestore-71da2e6e7e77 26. Can Level Order Traversal Of Binary Tree Be The Secret Weapon For Acing Your Next Interview - Verve AI, https://www.vervecopilot.com/interview-questions/can-level-order-traversal-of-binary-tree-be-the-secret-weapon-for-acing-your-next-interview 27. 10 Prompt Engineering Interview Questions and How To Prepare - Coursera, https://www.coursera.org/articles/prompt-engineering-interview-questions 28. Craft Powerful Success Stories with Promptitude | Transform Transcripts, https://www.promptitude.io/use-cases/success-story-creation