Architectural Specification and Implementation Strategy for "Obsidian Cortex": A Multi-Model, Agentic Knowledge Management System

1. Introduction: The Evolution of Agentic Personal Knowledge Management

The landscape of Personal Knowledge Management (PKM) is currently undergoing a fundamental paradigm shift, transitioning from passive repository systems—where data lies dormant until retrieved by the user—to active, agentic environments where artificial intelligence operates as a co-pilot, organizer, and synthesizer. The project proposed, herein referred to as "Obsidian Cortex," represents the pinnacle of this evolution. By integrating the local-first, highly extensible nature of Obsidian with the reasoning capabilities of state-of-the-art Large Language Models (LLMs) such as OpenAI's GPT-4o, Anthropic's Claude 3.5 Sonnet, and Google's Gemini 1.5 Pro, we aim to construct a system that does not merely answer questions but actively manages the user's digital cognition.

This report serves as a comprehensive architectural blueprint for developing Obsidian Cortex. It addresses the user's specific requirements for a tool that can view, read, create, modify, and organize the entirety of an Obsidian vault, while functioning as a personal assistant capable of interacting with the broader ecosystem of third-party plugins. The analysis is grounded in a deep technical understanding of the Obsidian TypeScript API, the comparative economics and capabilities of generative AI platforms, and the emerging field of local-first vector search databases suitable for Electron and mobile environments.

1.1 The Operational Mandate: "Organize Everything"

The core differentiator of Obsidian Cortex from existing AI plugins is its mandate to have full read/write access and the authority to organize the vault. Current tools typically operate in a "read-only" RAG (Retrieval-Augmented Generation) mode or create new notes in isolation. To fulfill the requirement of organizing "everything in my life," Obsidian Cortex must function as an operating system layer on top of the vault. This requires a rigorous implementation of file system abstractions that can reason about directory structures, taxonomy, and semantic relationships, effectively acting as an autonomous librarian that creates order from entropy.

1.2 The "NotebookLM" Standard: Grounding and Multimodality

The user has explicitly requested features akin to Google's NotebookLM. In the context of this

report, "NotebookLM features" are defined technically as:

1. **Source Grounding:** The ability to answer queries based *strictly* on the provided documents with zero hallucination, citing specific sources inline.[2]
2. **Multimodality:** The capacity to ingest and synthesize not just text, but images and audio, and to generate audio overviews (podcasts) of the content.[3]
3. **Deep Synthesis:** Moving beyond simple search to comprehensive thematic analysis across hundreds of documents.[4]

Replicating this within a local-first Obsidian environment requires a sophisticated hybrid architecture. While NotebookLM runs on Google's massive cloud infrastructure, Obsidian Cortex must achieve similar results using optimized local vector stores and intelligent API caching strategies to manage the computational load on user devices, particularly mobile phones.[5]

1.3 The Multi-Model Advantage

No single AI model currently dominates all performance metrics required for this project. By integrating three distinct API providers, Obsidian Cortex can implement a "Model Routing" strategy that optimizes for cost, speed, and capability:

- **Google Gemini 1.5 Pro:** With a context window of up to 2 million tokens and specific "Context Caching" capabilities, this model is the engine for "Whole Vault" analysis and the NotebookLM-style deep research tasks.[6]
- **Anthropic Claude 3.5 Sonnet:** Known for its superior nuance in writing and code generation, as well as "Prompt Caching," this model serves as the primary engine for drafting content and maintaining the "Personal Assistant" persona over long conversation histories.[8]
- **OpenAI GPT-4o:** With its robust function calling (tool use) benchmarks, this model acts as the "Executor," translating user intent into specific Obsidian commands and file operations.[9]

This report details how to weave these distinct threads into a cohesive, secure, and performant plugin.

---

2. The Obsidian Runtime Environment: Core API Architecture

To build a plugin that can "view, create, modify, and organize everything," one must first master the environment in which it lives. Obsidian is not a standard web application; it is a local-first application built on Electron (desktop) and Capacitor (mobile), which imposes strict

constraints and offers specific privileges regarding file access and process management.[1]

2.1 The App Interface and Global State

The entry point for any high-privilege interaction in Obsidian is the App interface. The plugin must treat this.app as the central nervous system, providing access to the Vault (storage), MetadataCache (indexing), Workspace (UI), and CommandManager (actions).[10]

For Obsidian Cortex, the App interface is not just a utility but the domain of operation. The plugin must be initialized with full access to this.app to perform system-wide operations. Unlike simple plugins that might only need access to the active leaf (the currently open file), Cortex requires a global view. This distinction is critical for the "Personal Assistant" requirement; if the user asks, "Where did I put that receipt from last week?", the agent cannot be limited to the current view—it must be able to scan the entire application state.

2.2 The Vault API: File System Abstraction and Concurrency

The Vault API is the mechanism through which the agent interacts with the user's data. To "organize everything," the agent must perform complex file operations that go beyond simple creation.

2.2.1 Reading Data: The Latency vs. Accuracy Trade-off

The plugin must implement a dual-strategy for reading files. The Vault API exposes read(file) and cachedRead(file).

- **read(file):** This reads directly from the disk. It is authoritative but slower.
- **cachedRead(file):** This reads from Obsidian's internal memory cache.

For the RAG (Retrieval-Augmented Generation) indexing process, which involves scanning thousands of notes, utilizing cachedRead() is mandatory to prevent I/O blocking that would freeze the interface. However, when the agent is performing a specific task, such as "edit the last paragraph of this note," it must use read() to ensure it isn't operating on stale data, particularly if the file is being synced externally (e.g., via iCloud or Obsidian Sync).

2.2.2 Modifying Data: Atomic Transactions with process()

A critical requirement is the ability to "modify" notes. A naive implementation using vault.read() followed by string manipulation and vault.modify() is dangerous. If the user types a character during the split-second the AI is thinking, the vault.modify() call will overwrite the user's new input.

To prevent data loss, Obsidian Cortex must strictly utilize Vault.process(file, callback). This method provides an atomic transaction: it reads the file, passes the content to the callback (where the AI's changes are applied), and then writes it back, handling any version conflicts that occurred in the interim. This is essential for an agent that operates in the background

while the user works.

2.2.3 Organization: The FileManager and Rename Logic

"Organizing everything" implies moving files. The Vault.rename(file, newPath) method is insufficient because it does not automatically update wikilinks pointing to that file. Obsidian Cortex must instead use FileManager.renameFile(file, newPath). This API ensures that if the agent moves "Project Alpha" to the "Archives" folder, every link to [[Project Alpha]] throughout the vault is automatically updated to [[Archives/Project Alpha]].[11] This distinction is vital to preserving the integrity of the user's knowledge graph.

2.3 The MetadataCache: The Semantic Skeleton

To function as a "Personal Assistant," the AI needs to understand the *structure* of the vault, not just the text. The MetadataCache API provides this structural intelligence without the heavy cost of reading file contents.

The cache contains resolved links, tags, frontmatter (YAML), and block references for every file.[12] Obsidian Cortex will utilize this to:

1. **Map Relationships:** When the user asks about a topic, the agent can check app.metadataCache.resolvedLinks to find all notes connected to that topic, traversing the graph to finding 2nd and 3rd order connections.
2. **Read Frontmatter:** The agent can quickly filter notes based on YAML properties (e.g., status: active or priority: high) without reading the full text, enabling rapid "organizing" actions like "Move all completed projects to the Archive folder".[13]

2.4 Mobile Environment Constraints

The user explicitly states, "I use Obsidian for everything in my life," implying significant usage on mobile devices. This introduces strict constraints:

- **No Node.js APIs:** Libraries like fs (File System) or child_process are unavailable on Obsidian Mobile (iOS/Android).[14] The plugin must rely *exclusively* on the Vault API for file operations.
- **Memory Limits:** Loading a local LLM or a heavy vector embedding model (like Transformers.js) can easily exceed the strict RAM limits of iOS apps, causing the OS to kill the Obsidian process.[15]
- **Recommendation:** The architectural decision here is to make "Local Embeddings" an optional setting, defaulting to "API-based Embeddings" (OpenAI/Gemini) on mobile devices to offload the computational heaviness to the cloud while maintaining the interface on the device.

---

3. The Multi-Model AI Engine: Cognitive Architecture

To meet the requirement of a "Personal Assistant" that integrates ChatGPT, Claude, and Gemini, Obsidian Cortex requires a sophisticated "Model Routing" layer. This layer acts as a traffic controller, analyzing the user's request and dispatching it to the model best suited for the specific cognitive task.

3.1 Comparative Economic and Performance Analysis

The choice of model is not just about capability but also about economic viability for a "personal" tool that sees heavy daily usage. The following table synthesizes the current capabilities and costs, derived from the research material.[6]

Model	Context Window	Caching Capability	Input Cost / 1M	Output Cost / 1M	Best Use Case for Cortex
Gemini 1.5 Pro	2,000,000 Tokens	**Context Caching:** Stores vast amounts of text (the whole vault) for repeated querying.	$1.25 (cached)	$5.00	"NotebookLM" deep research, Whole Vault Q&A.
Claude 3.5 Sonnet	200,000 Tokens	**Prompt Caching:** Caches system instructions and conversation history (ephemeral).	$3.00	$15.00	Complex reasoning, writing, coding, Persona maintenance.
GPT-4o	128,000 Tokens	Standard (No explicit caching	$2.50	$10.00	**Tool Use** (Function Calling), Real-time

		API).			Voice, Image Analysis.
Gemini Flash	1,000,000 Tokens	Context Caching available.	$0.075	$0.30	High-volume background tasks, tagging, quick summarization.

3.2 Gemini 1.5 Pro: The "Whole Vault" Brain

For features like "NotebookLM"—which implies deep understanding of massive datasets—Gemini's **Context Caching** is the architectural key. Standard RAG (Retrieval-Augmented Generation) chops documents into small chunks, losing the broader narrative arc. Gemini allows us to upload the *entire* relevant corpus (or even the whole vault, if under 2M tokens) into a cached state.[7]

Implementation Strategy:

1. **Cache Hydration:** When the user initiates a "Deep Research" session, Obsidian Cortex selects the relevant folders (or the whole vault). It concatenates the text content and uploads it to the cachedContents endpoint via the Gemini REST API.[17]
2. **TTL Management:** The cache is assigned a Time-To-Live (TTL). As long as the user keeps asking questions, the cache stays alive. This dramatically reduces cost ($1.25/1M vs standard rates) and latency, as the model doesn't need to re-read the documents for every follow-up question.[18]
3. **Use Case:** User asks, "Synthesize all my notes on Biology from the last three years into a coherent theory." Gemini Pro, with the full context loaded, can perform this synthesis far better than a chunk-based RAG system.

3.3 Anthropic Claude: The Persona Engine

The user wants a "Personal Assistant." This implies a consistent personality, knowledge of the user's preferences, and specific formatting rules. This "System Prompt" can grow very large (e.g., "Always use these specific headers," "Remember I am a visual learner," "Here is my bio").

Prompt Caching Strategy:
Sending this 5,000-token system prompt with every message is expensive and slow.

Anthropic's Prompt Caching allows Obsidian Cortex to mark the system prompt with cache_control: {"type": "ephemeral"}. The API caches this prefix.

- *Benefit:* Subsequent messages only pay for the *new* user query, not the massive system instruction block.[19]
- *Implementation:* The plugin will maintain a static "Assistant Persona" block at the start of every conversation thread, ensuring consistent behavior at a fraction of the cost.

3.4 OpenAI GPT-4o: The Agentic Executor

While Claude and Gemini excel at reading and writing, OpenAI's GPT-4o currently holds the edge in reliability for **Function Calling** (Tool Use). When the user says, "Move all notes about 'Taxes' to the Finance folder," the model must output a precise JSON object representing that command.

Streaming Tool Calls:
Obsidian Cortex will utilize OpenAI's streaming tool call capability. As the model decides to take an action (e.g., writing a long file), the plugin can stream this decision to the UI, showing a progress bar or a "Thinking..." indicator that updates in real-time, rather than waiting for the full generation to complete.9 This responsiveness is crucial for the "Assistant" feel.

---

4. Implementing "NotebookLM" Locally: The RAG & Grounding Subsystem

To fulfill the requirement of "features like NotebookLM," Obsidian Cortex must implement a local retrieval engine that supports source grounding, semantic search, and citation generation.

4.1 Vector Database Selection: The Local-First Imperative

Since the user uses Obsidian for "everything in my life," data privacy is paramount. Uploading the entire vault to a cloud vector database (like Pinecone) contradicts the local-first philosophy. The plugin must run the vector search on the user's device.

Technical Constraints:

- **Electron (Desktop):** Can run Node.js processes, allowing for robust databases.
- **Capacitor (Mobile):** Cannot run Python or binary executables easily; restricted to JavaScript/WASM.

The Solution: Orama
The research identifies Orama as the optimal solution for this hybrid environment.20

- **Architecture:** Orama is a vector search engine written entirely in TypeScript. It requires no native bindings, meaning it runs identically on macOS, Windows, iOS, and Android.[21]

- **Hybrid Search:** NotebookLM functionality requires finding *concepts* (vector search) and *specific keywords* (BM25). Orama supports hybrid search out-of-the-box, allowing the agent to find "Project Alpha" (keyword) even if the vector similarity is low, and "ideas about growth" (vector) where keywords don't match.[20]

4.2 Indexing Strategy: Chunking and Metadata

To support precise "inline citations" (e.g., ``), the vault cannot be indexed as whole files. It must be chunked.

The Indexing Pipeline:

1. **File Watcher:** The plugin listens to app.vault.on('modify'). When a note changes, it is re-indexed immediately.
2. **Semantic Chunking:** Instead of arbitrary fixed-size chunks (e.g., every 500 characters), the plugin should use Markdown-aware chunking. It splits text by Headers (#, ##) or double newlines. This ensures that a citation points to a distinct logical section of a note.[22]
3. **Metadata Extraction:** Each chunk stored in Orama includes:
   - content: The text vector.
   - filePath: The source note path.
   - blockId: A generated block reference ID (e.g., ^8f9a2b) attached to the paragraph in Obsidian. This enables the "Deep Link" capability where clicking a citation scrolls exactly to the relevant text.[23]

4.3 The Grounding Algorithm: Zero-Hallucination Citations

"Grounding" is the process of forcing the AI to only use the retrieved context. This is achieved via a specific Prompt Engineering pattern combined with the RAG retrieval.

The "Citation" Prompt Template:
You are a research assistant. Answer the user's question using ONLY the provided Context snippets.
Rules:

1. You strictly cannot use outside knowledge.
2. Every claim must be cited using the format [[File Name#^blockid]].
3. If the answer is not in the context, state "I cannot find this information in your notes."

Context:
-: "Mitochondria is the powerhouse..."
UI Rendering:
When the LLM outputs], the Obsidian Cortex UI (built in React) intercepts this string. Instead of displaying the raw link, it renders a clickable footnote badge (e.g., ``). Hovering over this badge triggers a popover showing the source text, mimicking the NotebookLM UI interaction.2

4.4 Audio Synthesis: The Podcast Feature

NotebookLM's viral feature is the "Audio Overview." Obsidian Cortex will replicate this using the OpenAI or Gemini Audio APIs.

Implementation:

1. **Script Generation:** The user selects a folder or set of notes. The Agent uses Claude 3.5 Sonnet to "Generate a two-person podcast script discussing these notes, with a host and a guest."
2. **Speech Synthesis:** The script is parsed. "Host" lines are sent to OpenAI gpt-4o-mini-tts using the alloy voice; "Guest" lines use the echo voice.[24]
3. **Playback:** The resulting audio buffers are concatenated (using a library like ffmpeg.wasm or simple Buffer concatenation) and played via an HTML5 audio player embedded in the note or the plugin sidebar.

---

5. Agentic Capabilities: Interacting with the Plugin Ecosystem

The most ambitious requirement is for the tool to "interact with any and all other plug-ins commands." This transforms Obsidian Cortex from a chatbot into an Agentic Operating System.

5.1 The Command Registry Discovery Mechanism

Obsidian's internal command registry is accessible via this.app.commands. This registry contains every action available in the app, from core features ("Toggle Dark Mode") to third-party plugin actions ("Kanban: Create new board").

Discovery Tool:
To allow the AI to "know" what it can do, the plugin implements a tool named list_available_commands.

- **Logic:** It iterates through this.app.commands.listCommands(), extracting the id and name of each command.[25]
- **Filtering:** To prevent context window overflow, it performs a fuzzy search on the command names based on the user's intent. If the user says "Create a task," the tool returns commands containing "task," "todo," "create," etc.

5.2 The Execution Tool

Once the AI identifies the correct command (e.g., tasks:create-task), it utilizes the execute_command tool.

- **Implementation:** The tool calls

this.app.commands.executeCommandById(commandId).[26]

Contextual Awareness Risk:
Many Obsidian commands only work if a specific type of view is active (e.g., "Insert Template" requires a Markdown Editor to be focused). If the AI tries to run this command while looking at the Graph View, it will fail.

- **Mitigation:** The execute_command tool must include logic to check app.workspace.getActiveViewOfType(MarkdownView). If no editor is active, the Agent must first trigger the open_file action to ensure a valid context exists before executing the command.[25]

5.3 The ReAct (Reasoning + Acting) Loop

To "Organize Everything," the agent needs to perform multi-step workflows. This requires a **ReAct** architecture.

Example Scenario: "Find all notes about 'Gardening', tag them with #hobby, and move them to the 'Personal' folder."

Agent Loop:

1. **Thought:** I need to find the notes first.
2. **Action:** Call search_vault("Gardening").
3. **Observation:** Returns list: ``.
4. **Thought:** Now I need to modify these files to add the tag.
5. **Action:** Call modify_note("Rose Care.md", "append_frontmatter", "tags: hobby").
6. **Action:** Call modify_note("Soil pH.md", "append_frontmatter", "tags: hobby").
7. **Thought:** Now I need to move them.
8. **Action:** Call move_note("Rose Care.md", "Personal/Rose Care.md").
9. **Action:** Call move_note("Soil pH.md", "Personal/Soil pH.md").
10. **Final Response:** "I have tagged and moved 2 notes to the Personal folder."

This recursive loop allows the agent to handle complex, ambiguous instructions that a simple "one-shot" command could not.[27]

---

6. Security, Storage, and Cross-Platform Engineering

Given the high-privilege access ("Full access to Obsidian notes") and the sensitivity of the data, the security architecture must be robust.

6.1 Secure Key Management: The safeStorage Dilemma

The plugin requires API keys for OpenAI, Anthropic, and Google. Storing these in plain text in

the data.json file is a security risk, as this file is often synced unencrypted via Dropbox or Git.

Desktop Strategy (Electron):
On macOS, Windows, and Linux, the plugin will leverage Electron's safeStorage API.

- **Mechanism:** This API encrypts data using the OS-level keychain (e.g., Apple Keychain, Windows DPAPI). The key is never stored on disk in a way that can be read by other applications or synced to the cloud.[28]
- **Code Path:** window.require('electron').remote.safeStorage.encryptString(apiKey).

Mobile Strategy (Capacitor):
The safeStorage API is not available on mobile. This presents a critical challenge.

- **Solution:** The plugin must implement a fallback encryption using a user-provided passphrase or a device-specific local salt stored in localStorage (which is sandboxed to the app). While less secure than hardware-backed encryption, it protects the key from being exposed if the vault files are leaked.[29]
- **Sync Warning:** The report explicitly recommends *not* syncing API keys via Obsidian Sync. Users should be prompted to re-enter keys on each device to ensure they remain in local, secure storage.

6.2 Data Privacy and Egress Control

To respect the "Personal" nature of the assistant, the plugin should implement a "Privacy Mode."

- **Local Indexing:** All vector indexing (via Orama) happens locally. The text content is only sent to the embedding API (OpenAI) if explicitly enabled. If "Privacy Mode" is on, the plugin should fallback to a local embedding model (like Xenova/all-MiniLM-L6-v2 running in Transformers.js), keeping the entire indexing pipeline offline.[30]
- **Explicit Consent:** Before sending the "Whole Vault" to Gemini for Context Caching, the UI must display a clear confirmation dialog: "This action will upload 1.5MB of text to Google's API for processing. Do you consent?"

6.3 Mobile Optimization

Running a full Agentic RAG system on a phone requires aggressive optimization.

1. **Lazy Loading:** The Orama index should not load into memory until the user opens the AI interface.
2. **Offloading:** On mobile, local embedding generation (using the device CPU) drains battery and risks crashing the app. The architecture dictates using **API-based embeddings** (OpenAI text-embedding-3-small) on mobile to shift the compute load to the cloud, ensuring the UI remains responsive.[15]

---

7. Future Outlook and Maintenance

The architecture defined here is built for the current state of the art (late 2025). However, the AI field moves rapidly.

- **Model Agnosticism:** The "Model Routing" layer is designed as an interface (IAIProvider). Adding a future model (e.g., Llama 4 local) effectively requires just adding a new class implementing this interface, ensuring the plugin remains future-proof.
- **MCP Integration:** The architecture aligns with the emerging "Model Context Protocol" (MCP). Future versions could expose the Obsidian Vault as an MCP Server, allowing *external* agents (like Claude Desktop) to query the vault without the plugin needing to be the active interface.[27]

---

8. Technical Addendum: Data & Schemas

8.1 API Capabilities Matrix

The following table details the specific API capabilities mapped to the user's requirements.

User Requirement	Technical Implementation	API/Library
"View/Read"	Vault.cachedRead(), MetadataCache	Obsidian Core API
"Create/Modify"	Vault.process(), FileManager.renameFile()	Obsidian Core API
"Organize Everything"	ReAct Loop + app.fileManager + app.vault	Agentic Logic
"NotebookLM Features"	Orama Vector Search + Citation Prompting + TTS	@orama/orama, OpenAI Audio
"Interact with plugins"	app.commands.executeCommandById()	Obsidian Command API
"Use on Mobile"	Capacitor-safe logic, Cloud Embedding fallback	Cross-platform JS

8.2 JSON Schema for Tool Definitions

To ensure the LLM interacts correctly with the vault, the following JSON schemas must be passed to the tools parameter of the API.

Create Note Tool:

JSON

```
{
  "name": "create_note",
  "description": "Create a new markdown note in the vault.",
  "parameters": {
    "type": "object",
    "properties": {
      "path": {
        "type": "string",
        "description": "Full path including folder and.md extension (e.g., 'Projects/Alpha.md')"
      },
      "content": {
        "type": "string",
        "description": "The markdown content of the note."
      }
    },
    "required": ["path", "content"]
  }
}
```

Execute Command Tool:

JSON

```
{
  "name": "execute_command",
  "description": "Execute an Obsidian command (core or plugin).",
  "parameters": {
```

```
    "type": "object",
    "properties": {
      "command_id": {
        "type": "string",
        "description": "The specific ID of the command (e.g., 'editor:toggle-bold')"
      }
    },
    "required": ["command_id"]
  }
}
```

9. Conclusion

Obsidian Cortex represents a significant leap forward in local-first knowledge management. By adhering to this architectural specification, the resulting plugin will provide a secure, highly capable, and truly agentic experience. It respects the user's privacy through local vector stores and secure key management, while leveraging the immense power of cloud-based LLMs for reasoning and synthesis. This is not merely a plugin; it is a foundational layer for the next generation of thought processing.

Works cited

1. Vault - Developer Documentation, accessed December 10, 2025, https://docs.obsidian.md/Plugins/Vault
2. NotebookLM: Document-Grounded AI by Google - Emergent Mind, accessed December 10, 2025, https://www.emergentmind.com/topics/notebooklm
3. The Ultimate Guide to Google NotebookLM | Features, Setup & Use Cases - Deimos Cloud, accessed December 10, 2025, https://www.deimos.io/blog-posts/the-ultimate-guide-to-google-notebooklm
4. AI Literature Reviews: Exploring Google's NotebookLM for Analysing Academic Literature, accessed December 10, 2025, https://broneager.com/ai-literature-review-notebooklm
5. Obsidian for iOS and Android, accessed December 10, 2025, https://obsidian.md/mobile
6. OpenAI API vs Anthropic API vs Gemini API: A practical guide for businesses in 2025, accessed December 10, 2025, https://www.eesel.ai/blog/openai-api-vs-anthropic-api-vs-gemini-api
7. Caching | Gemini API - Google AI for Developers, accessed December 10, 2025, https://ai.google.dev/api/caching
8. Prompt Caching - Portkey Docs, accessed December 10, 2025, https://portkey.ai/docs/integrations/llms/anthropic/prompt-caching
9. Using tools | OpenAI API, accessed December 10, 2025, https://platform.openai.com/docs/guides/tools
10. obsidian-api/README.md at master - GitHub, accessed December 10, 2025,

https://github.com/obsidianmd/obsidian-api/blob/master/README.md
11. App - Developer Documentation - Obsidian Developer Docs, accessed December 10, 2025, https://docs.obsidian.md/Reference/TypeScript+API/App
12. MetadataCache - Developer Documentation, accessed December 10, 2025, https://docs.obsidian.md/Reference/TypeScript+API/MetadataCache
13. Properties - Obsidian Help, accessed December 10, 2025, https://help.obsidian.md/properties
14. Hacking Obsidian: Making a Desktop-only Plugin Work on your Phone or Tablet - Medium, accessed December 10, 2025, https://medium.com/obsidian-observer/hacking-obsidian-making-a-desktop-only-plugin-work-on-your-phone-or-tablet-d49ed357b02c
15. [AINews] 12/28/2023: Smol Talk updates - Buttondown, accessed December 10, 2025, https://buttondown.com/ainews/archive/ainews-12282023-smol-talk-updates/
16. LLM API Pricing Comparison (2025): OpenAI, Gemini, Claude | IntuitionLabs, accessed December 10, 2025, https://intuitionlabs.ai/articles/llm-api-pricing-comparison-2025
17. Context caching | Gemini API | Google AI for Developers, accessed December 10, 2025, https://ai.google.dev/gemini-api/docs/caching
18. Lowering Your Gemini API Bill: A Guide to Context Caching | by Raheel Siddiqui | Medium, accessed December 10, 2025, https://rawheel.medium.com/lowering-your-gemini-api-bill-a-guide-to-context-caching-0e1f4d0cb3f8
19. Prompt caching - Claude Docs, accessed December 10, 2025, https://platform.claude.com/docs/en/build-with-claude/prompt-caching
20. OramaSearch: Your New Favorite Search Engine - Codemotion, accessed December 10, 2025, https://www.codemotion.com/magazine/backend/oramasearch-your-new-favorite-search-engine/
21. Orama - GitHub, accessed December 10, 2025, https://github.com/oramasearch
22. Obsidian plugin for Vector Search, accessed December 10, 2025, https://www.obsidianstats.com/plugins/vector-search
23. Build RAG with in-line citations | LlamaIndex Python Documentation, accessed December 10, 2025, https://developers.llamaindex.ai/python/examples/workflow/citation_query_engine/
24. Text to speech | OpenAI API, accessed December 10, 2025, https://platform.openai.com/docs/guides/text-to-speech
25. Commands - Developer Documentation, accessed December 10, 2025, https://docs.obsidian.md/Plugins/User+interface/Commands
26. Triggering an Obsidian command from within an event callback - Developers: Plugin & API, accessed December 10, 2025, https://forum.obsidian.md/t/triggering-an-obsidian-command-from-within-an-event-callback/37158
27. Tool use with Claude - Claude Docs, accessed December 10, 2025,

https://platform.claude.com/docs/en/agents-and-tools/tool-use/overview
28. safeStorage | Electron, accessed December 10, 2025, https://electronjs.org/docs/latest/api/safe-storage
29. Cross-platform secure storage for secrets and tokens that can be sync'd - Obsidian Forum, accessed December 10, 2025, https://forum.obsidian.md/t/cross-platform-secure-storage-for-secrets-and-tokens-that-can-be-syncd/100716
30. Local JavaScript Vector Database that works offline - RxDB, accessed December 10, 2025, https://rxdb.info/articles/javascript-vector-database.html