Meeting Summarizer Agent with Google Workspace Integration

Jyotika Kakar
jkakar@ucsd.edu

Kanaad Deshpande
kadeshpande@ucsd.edu

Krishi Chawda
kchawda@ucsd.edu

Tanmay Nale
tnale@ucsd.edu

1 Introduction

Teams spend a significant amount of time in meetings, yet the work that follows documenting decisions, tracking action items and scheduling next steps, remains largely manual. This manual overhead creates a productivity bottleneck leading to the need for automation of post-meeting workflows.

A key challenge in this process is that meeting information naturally accumulates context over time: decisions, updates, and deadlines evolve across multiple sessions. Agents that operate on isolated transcripts without context cannot retain or reason over long-term information, leading to fragmented understanding and follow-ups.

To address this challenge, we built a Meeting Summarizer Agent with multi-tool Google Workspace integration that analyzes meeting transcripts, extracts actionable insights and automates follow-ups through Google Calendar and Google Tasks. The agent leverages Google Gemini for intelligent summarization and a context-aware memory mechanism that enables information to be preserved and propagated across meetings and tool interactions.

2 Project Evolution

2.1 Phase 1: Context-Free Meeting Summarization

We first developed a context-free meeting summarization agent that leverages Google's Gemini to analyze individual meeting transcripts and extract actionable insights. Given a raw transcript, the agent produces a TL;DR summary, identifies key decisions, extracts action items with due dates, highlights blockers and captures the main discussion points.

To streamline post-meeting workflows, the agent integrates with Google Workspace, automatically creating tasks in Google Tasks and scheduling follow-up events in Google Calendar. The system exposes a RESTful API built with Flask, supporting both direct transcript submission and retrieval from AWS S3. We implemented multi-user session management to track analysis history and runtime metrics such as request latency, and containerized the system using Docker for deployment on AWS EC2.

While this initial system established a robust summarization and tool-integration pipeline, each meeting was processed in isolation without historical context, preventing the agent from leveraging prior decisions or evolving discussion threads. This limitation directly motivated the extension toward persistent storage and cross-meeting context awareness.

2.2 Phase 2: Context-Aware Meeting Summarization

Building on the foundation established in Phase 1, we extended the agent with persistent context management to address the limitation of isolated meeting processing. In this phase, context encompasses all relevant information from past and current user interactions that influence system outputs, including meeting transcripts, decisions and action items, follow-up meetings, user preferences and cross-meeting connections that establish continuity by linking current discussions to previous meetings.

To enable context-aware processing, we introduced **MCP servers** that retrieve meeting transcripts from the file system and query the SQLite database for historical meeting data. This architecture allows the agent to reference prior summaries, decisions and action items when analyzing new sessions, transforming the system from a stateless processor into a memory-enabled assistant. We implemented *thread-based context isolation* that treats meeting series as connected narratives-each thread maintains its own context boundary, enabling the agent to track project-specific discussions while supporting **cross-user context sharing** within the **same thread**.

This context management infrastructure enabled several key capabilities: tracking progress on action items across meetings, understanding how timelines and budgets evolve, detecting duplicate tasks and explicitly connecting current discussions to prior outcomes through intelligent cross-referencing captured in a dedicated *context_connections* field. The result is a system that maintains narrative continuity across the entire lifecycle of a project, rather than treating each meeting as a disconnected event.

3 Novelty

Our Meeting Summarizer Agent introduces several novel contributions that distinguish it from existing meeting automation tools. The core innovation lies in how we architect

context management and tool integration as explained:

3.1 Thread-Based Context with Cross-User Sharing:

A key architectural decision is our implementation of thread-based context isolation that treats meeting series as connected narratives rather than isolated events. This allows the system to track progress, understand evolving decisions and identify dependencies across multiple sessions-something traditional meeting tools fail to do as they operate on single transcripts in isolation. Going beyond single-user systems, our architecture supports *cross-user context sharing*, allowing different users within the same thread to benefit from accumulated meeting history and insights, enabling true collaborative intelligence.

3.2 MCP-First Architecture for Context Management:

The system's primary novelty is leveraging the Model Context Protocol (MCP) as a foundational framework for both tool integration and cross-meeting context management. Unlike traditional point-to-point API integrations that require custom implementations for each service, MCP provides a *unified interface* across Google Calendar, Google Tasks and the local database. This enables enhanced composability through intelligent tool chaining after extracting action items from transcripts, the agent automatically routes them to Google Tasks, schedules follow-up meetings in Calendar and updates the database, all while maintaining full context about source meetings, owners and dependencies throughout the entire workflow. This standardized approach not only ensures consistent data flow but also allows seamless addition of new tools without modifying core agent logic, making the system highly extensible.

3.3 Persistent Relational Storage for Multi-Meeting Intelligence:

We employ a normalized SQL database architecture that, when combined with MCP's context passing capabilities, enables the agent to maintain cross-meeting intelligence recognizing completed tasks, tracking budget evolution across sessions and detecting duplicate action items. This transforms fragmented meeting data into coherent project narratives allowing the system to understand not just what was discussed, but how commitments and decisions evolved over time.

4 Technical Implementation Details

4.1 System Architecture and Core Dependencies

The Meeting Summarizer Agent is implemented as a Python-based, context-aware LLM inference system following a layered architecture with four primary components: input processing, AI inference, persistent storage and external service integration.

For each request, the system constructs a prompt by combining the current meeting transcript with relevant historical artifacts retrieved from prior meetings. This prompt is submitted to the Gemini large language model, which emits a structured JSON representation encoding summaries, decisions, action items, risks and discussion topics. This enables cross-meeting context propagation.

All extracted outputs are persisted in a SQLite-backed relational store, which serves as the system's externalized memory layer. Retrieved records are selectively re-injected into subsequent inference calls to maintain longitudinal context.

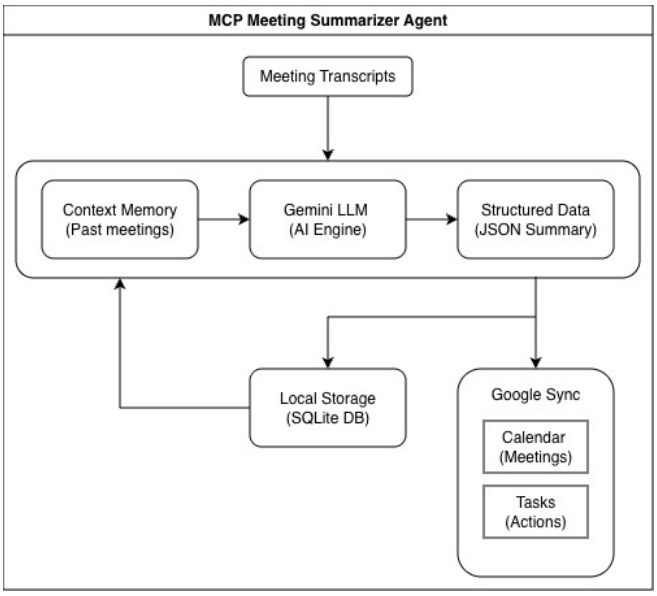

Figure 1: Architecture Diagram

The structured outputs are consumed by a Google Workspace execution layer which materializes actions through the Google Calendar and Google Tasks APIs, forming a closed loop between LLM inference, persistent memory, and downstream tool execution.

The system relies on three core dependencies: the Google Generative AI SDK for LLM access, the Google API Client for Calendar and Tasks integration and SQLite3 for relational persistence. The central orchestration unit, MCPMeetingAgent, supports thread-based user isolation, cross-user context sharing and optional service enablement through dependency injection.

4.2 Context Management and Memory System

The memory subsystem follows a three-tier hierarchical design with distinct persistence characteristics and retrieval semantics. Figure 2 illustrates the Cross-tool Context Flow showing cohesive workflow from transcript analysis to external productivity tools.

Intra-session memory is maintained through instance variables within the agent, including cumulative latency metrics, request counters and live database connection handles. This ephemeral state exists only for the lifetime of the Python

process and supports runtime observability and performance tracking.

Inter-session persistence is implemented using SQLite with a normalized relational schema. The primary **meetings** table stores records indexed by a thread identifier for *user isolation, ISO 8601 timestamps, executive summaries* and full extraction results serialized as *JSON blobs*. Supporting tables for **action_items** and **decisions** maintain foreign-key relationships with the parent meeting record, enabling efficient querying of outstanding tasks and historical decisions. A separate **calendar_events** table tracks synchronization state by storing Google-assigned resource identifiers alongside local references.

Context retrieval is performed through the `get_context_from_db()` method, which executes parameterized, thread-scoped queries to retrieve the meeting summaries, outstanding action items and cross-user summaries from the global thread namespace. Cross-user results exclude the current user's contributions to prevent redundant context injection. The retrieved records are formatted into a structured natural-language block suitable for prompt augmentation.

Cross-user memory sharing is enabled through a dual-write strategy. When persisting meeting results using `store_meeting_in_db()`, the system writes both a full local record and a condensed summary into the global thread namespace, prefixed with source user identifier using bracket notation. This mechanism enables team-wide context awareness while preserving strict attribution.

4.3 Transcript Processing Pipeline

The extraction pipeline is powered by Google's Gemini 2.0 Flash model **gemini-2.0-flash**, selected for its favorable trade-off between inference latency and output quality. Model invocation is encapsulated in the `_call_gemini()` method, which manages API communication and propagates structured error states.

The primary processing workflow is implemented in the `summarize()` method. Prompt construction begins with conditional context injection; when historical data is present, the system prepends prior meeting summaries and unresolved action items to the current transcript. The model is explicitly instructed to identify cross-meeting dependencies and to attribute referenced content using source user tags.

The prompt enforces a structured JSON output schema consisting of six extraction categories: (i) an executive summary, (ii) a decisions array with ownership and rationale, (iii) an action items array with assignees and due dates, (iv) a meetings-to-schedule array, (v) a risks array and (vi) a context connections array linking to prior records.

Response parsing employs cascading fault-tolerant logic. The primary path removes markdown code fencing where present, while a secondary path uses regular-expression-based extraction to isolate malformed JSON. Default values are enforced via `setdefault()` operations to ensure schema completeness under partial model output.

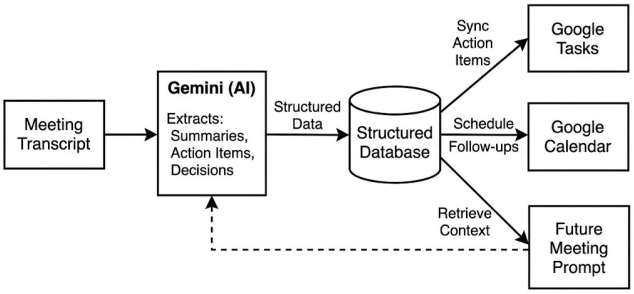

Figure 2: Cross-tool Context Flow Diagram

4.3.1 Context Prioritization for Memory Injection

Contextual signals differ in their impact on agent performance and are therefore prioritized during memory injection. *Action item history* (highest priority) provides critical operational continuity by preserving unresolved tasks and deferred commitments across sessions. *Meeting summary history* (high priority) enables tracking of recurring agenda items and sustained discussion threads. *Decision records* (medium priority) capture directional commitments, ownership and rationale, supporting longitudinal reasoning over project evolution. *Raw transcript data* (low priority) is retained primarily for offline analysis and model refinement and is not required for real-time reasoning once structured artifacts are extracted.

4.4 External Service Integration

All third-party API interactions are encapsulated within the **GoogleIntegration** class, which initializes Google Calendar and Google Tasks service clients at runtime. Authentication is implemented using the **OAuth 2.0** authorization code flow with automatic token refresh. For first-time authorization, a local authorization server is launched to securely acquire and persist user credentials.

Task synchronization is performed through the `create_task()` method, which maps extracted action items into Google Tasks resources while preserving ownership metadata and normalizing due-date representations.

Calendar event creation employs a conflict-aware scheduling strategy based on availability checks and iterative free-slot discovery within configurable business hours. The scheduling mechanism prioritizes the preferred time window and escalates to the next available slot upon conflict detection.

5 Baseline Evaluation & Findings

5.1 Evaluation Methodology

For baseline evaluation, we created a dataset of 18 meeting transcripts simulating realistic scenarios including team

stand-ups, project reviews, and planning sessions. Each transcript was manually annotated by two team members to establish ground truth for summaries, action items, decisions, and risks, with disagreements resolved through discussion.

We evaluated the system in two phases: Phase 1 (meetings 1-9) with context disabled, and Phase 2 (meetings 10-18) with context enabled after the first meeting in each thread. Performance metrics were computed by comparing system outputs against ground truth annotations, with a match requiring semantic equivalence rather than exact text matching. Three representative samples (Meetings 14, 15, and 18) spanning varied complexity levels were selected for detailed qualitative analysis.

5.2 Quantitative Performance

Refer table 1.

5.3 Qualitative Observations

5.3.1 Strengths

The agent demonstrates strong performance in single-session analysis, consistently producing concise and well-structured outputs. It is particularly effective at extracting implicit action items and correctly attributing task ownership, even when responsibilities are inferred rather than explicitly stated. The model reliably parses multiple due dates within a single transcript while maintaining contextual coherence. Decision points and risk factors are documented with high fidelity. Integration with Google Tasks and Calendar is seamless, enabling reliable end-to-end automation. Finally, the system generates clean, consistent JSON outputs, enabling deterministic downstream processing.

5.3.2 Limitations

In multi-meeting simulations, the agent exhibits limited continuity awareness, frequently failing to recognize repeated agenda items and generating redundant follow-ups instead of referencing prior outcomes. Due to the absence of long-term semantic understanding, the system cannot reliably distinguish between newly introduced tasks and ongoing commitments. Follow-up meeting detection from transcript content remains unreliable, with a nearly 0% success rate observed during evaluation. Due date normalization also degrades under informal or ambiguous natural-language expressions. Finally, the fixed context window constrained to the most recent n meetings may omit historically significant information required for long-term project understanding.

5.4 Statistical Analysis

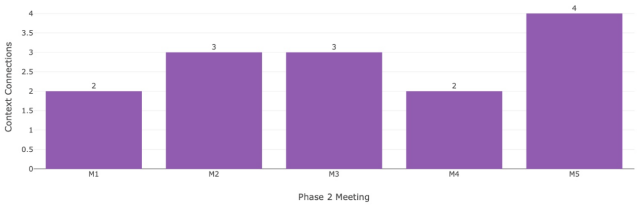

Figure 3: Context Connections in Phase 2 Meetings

Figure 3 shows that meetings in Phase 2 utilize context from previous meetings, as seen by the presence of context connections across all five meetings. Each meeting shows between 2 to 4 context connections, confirming that the context-aware functionality is working as intended.

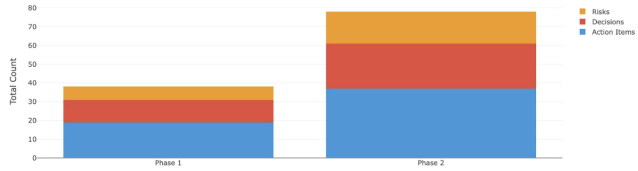

Figure 4: Total Insights by Phase

The stacked bar chart in Figure 4 compares insights from Phase 1 (without context) versus Phase 2 (with context) across Action Items, Decisions & Risks. Total insights more than double, increasing from nearly 38 to 78.

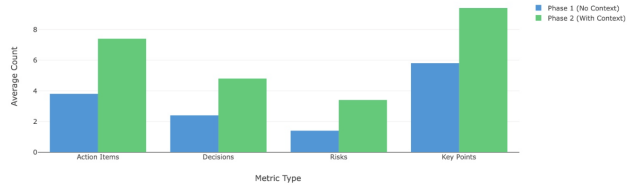

Figure 5: Average Metrics per Meeting

As seen in Figure 5, all categories grew, with Action Items remaining the largest. The increase is driven by context-aware processing, allowing the system to track unfinished tasks, identify cross-meeting dependencies, break down vague tasks into actionable steps, and generate follow-up items based on prior discussions. This demonstrates that providing historical and conversational context significantly improves the extraction of actionable insights from meeting transcripts.

Metric	Performance	Details
Summary Generation	88.9% (8/9)	Most summaries accurate and concise; minor omissions in longer transcripts
Action Item Extraction	85.7% (12/14)	Missed some implicit or multi-step tasks; handled explicit ones well
Decision Tracking	77.8% (7/9)	Some decisions captured without full context or rationale
Risk Identification	66.7% (2/3)	Detected clear risks but missed subtler ones in longer meetings
Google Tasks Creation	85.7% (12/14)	Most action items converted successfully; some due dates missing
Primary Calendar Event	100% (3/3)	Created one main event for each meeting as expected

Table 1: Quantitative Performance

6 Output

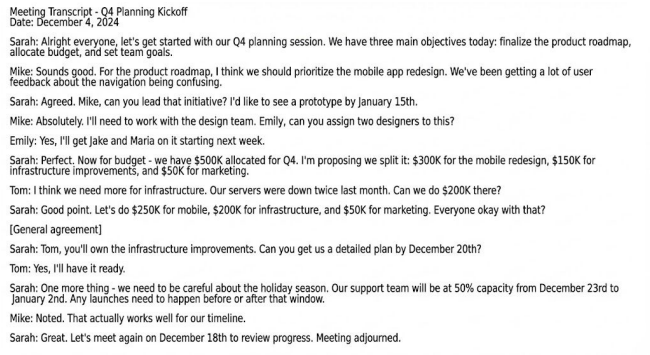

Figure 6: Meeting transcripts without any prior context

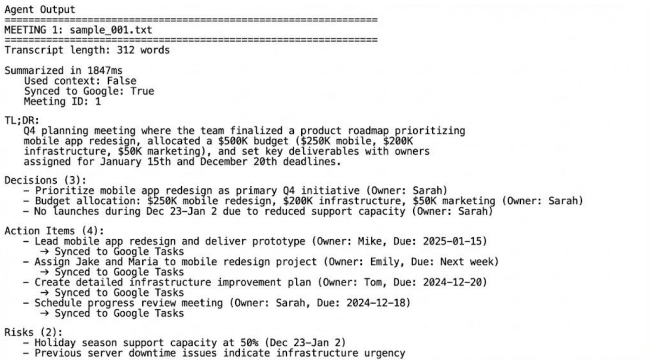

Figure 7: Meeting output

6.1 Key Advantages of Context in Meeting 2

6.1.1 Progress Tracking & Accountability

Figure 7 shows that **without context**, the system only extracts new action items without tracking prior task completion or ownership continuity. Figure 9 demonstrates that **with context**, the system identifies completed assignments (Emily assigning Jake and Maria), verifies deadline compliance (Tom's December 20 deliverable), tracks ongoing progress (Mike's mobile redesign), and distinguishes fulfilled from pending responsibilities.

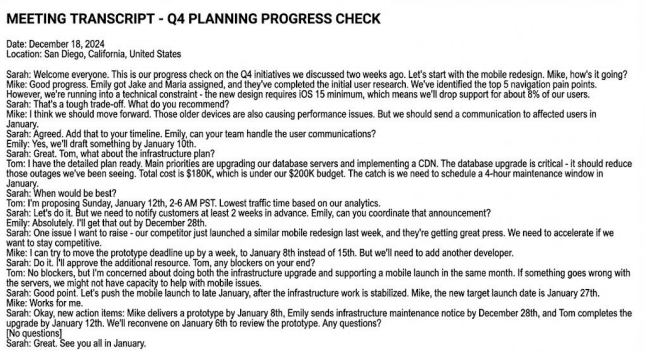

Figure 8: Meeting transcript with prior context

6.1.2 Budget & Resource Management

Figure 7 shows that **without context**, the system treats budget values as isolated numerical figures ($250K for mobile and $200K for infrastructure), with no ability to associate them with subsequent execution or fiscal validation. **With context** (Figure 9), the system correctly identifies that Tom's finalized plan costs $180K against the originally allocated $200K budget, indicating the project remains within approved financial limits. The generated context connection explicitly records *"Budget allocation staying within approved limits,"* enabling stakeholders to assess fiscal responsibility and budget adherence across meetings.

6.1.3 Timeline Evolution & Decision Rationale

Without context (Figure 7), dates are processed as isolated facts. **With context** (Figure 9), the system tracks timeline evolution, recognizing that the Jan 8^{th} deadline accelerates the original Jan 15^{th} target and that the Jan 27^{th} launch represents a delay. It captures underlying rationale-competitive pressure and infrastructure conflicts-and identifies decision interdependencies, such as the constraint preventing launch during maintenance windows.

6.1.4 Intelligent Context Connections

This is the **most powerful** feature: Context Connections (3):

- Mobile app redesign progress on schedule from Meet 1.
  Reference: Mike's assigned initiative from Q4 plan meet.
- Infrastructure improvements plan completed as requested.
  Reference: Tom's Dec 20^{th} deliverable from previous meet.

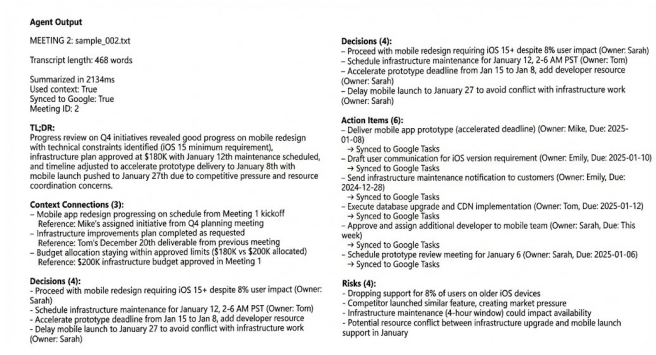

Figure 9: Meeting output showing context

- Budget allocation staying within approved limits ($180K vs $200K allocated). Reference: $200K infrastructure budget approved in Meeting 1.

6.1.5 Smarter Google Task Creation

Without context, the system creates tasks in isolation with no connection to previous commitments. **With context** (see Figures 10 and 11, task notes reference broader context (e.g., "accelerated timeline due to competition"), the system distinguishes continuing tasks from new initiatives, and can flag incomplete tasks from prior meetings.

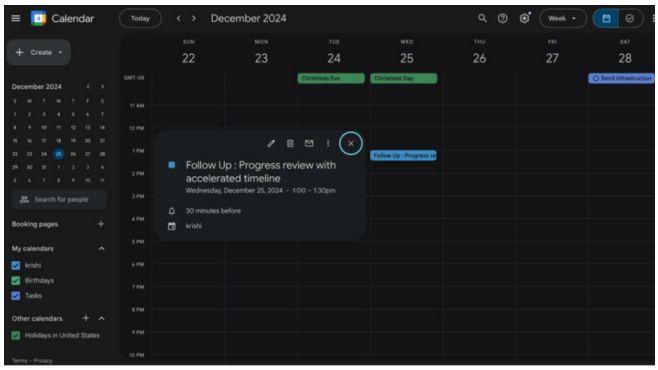

Figure 10: Google Task

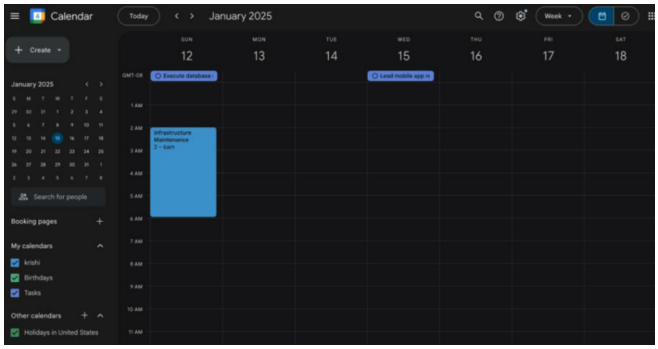

Figure 11: Google Calendar

6.1.6 Executive Summary Quality

The contextual TL;DR summary does exhibit narrative coherence by capturing evolving project dynamics, external pressures and scheduling adjustments rather than presenting only a static description of meeting topics.

7 Key Achievements

1. Cross-User Context Sharing: Enables controlled team-wide knowledge sharing through a global thread mechanism for collaborative awareness.
2. Context-Aware Meeting Analysis: Persistent inter-session memory allows the system to link current meetings with historical decisions and unresolved action items.
3. Multi-Layer Memory Architecture: Implements a three-tier memory hierarchy (intra-session, inter-session, and cross-user) using lightweight local storage.
4. Structured Information Extraction: Unstructured transcripts transformed into actionable structured outputs including summaries, decisions, action items, and risks.
5. Intelligent Calendar Scheduling: Conflict-aware scheduling with automatic fallback to available time slots.
6. Bidirectional Productivity Tool Integration: Synchronizes extracted outcomes directly into Google Tasks and Google Calendar for full post-meeting automation.

8 Limitations and Future Work

The current system relies on chronological context retrieval without semantic search, which may reduce retrieval precision in long-running or complex projects. Future work will integrate vector embedding–based semantic retrieval with retrieval-augmented generation (RAG) to dynamically select relevant historical context. The system also lacks deduplication mechanisms for overlapping meeting items and provides limited support for long-term project evolution. Future enhancements will address semantic deduplication and long-horizon project tracking through richer temporal analytics.

9 Conclusion

The Meeting Summarizer Agent demonstrates strong performance as a stateful summarization and workflow automation system with persistent storage and Google Workspace integration. The SQLite database provides a foundation for context management, and the Google API integration creates a seamless workflow from transcript to actionable tasks.

10 Resources and Relevant Works

- GitHub Repository: https://github.com/Jyotikakakar/Briefly-a-meeting-summarizer-agent.
- AMI Meeting Dataset: https://groups.inf.ed.ac.uk/ami/corpus/
- Building MCP servers: https://platform.openai.com/docs/mcp
- Google APIs: Calendar API v3, Tasks API v1
- AI Model: Google Gemini 2.0 Flash