International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 3, May-June 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Mavinmail: An Intelligent Email Management System Using Large Language Models and Retrieval-Augmented Generation

Ronak Sanjay Rahane, Vyankatesh Ratnakar Kamod, Prathmesh Ravindra Patil,
Gargee Shekhar Buva, Vrushali S Nikam

Department of Computer Engineering, Gokhale Education Society's R. H. Sapat College of Engineering, Management Studies and Research, Savitribai Phule Pune University, Pune, Maharashtra, India

Emails: 22_ronak.rahane@ges-coengg.org, 22_vyankatesh.kamod@ges-coengg.org, 22_prathmesh.patil@ges-coengg.org, 22_gargee.buva@ges-coengg.org, vrushali.nikam@ges-coengg.org

Abstract:

The exponential growth of digital communication has positioned email as the primary medium for professional correspondence, simultaneously imposing severe cognitive load and information overload on modern workers. This paper presents Mavinmail, an advanced cross-platform email assistant engineered to automate email processing and contextual summarization. Architected as a cloud-native monorepo utilizing Turborepo, Mavinmail bridges a React-based browser extension and a Next.js dashboard through a unified Node.js/Express backend. To achieve latency-resistant, high-accuracy context generation, the system employs a Retrieval-Augmented Generation (RAG) pipeline powered by Pinecone Serverless vector indexes and Cohere embeddings, routing generation tasks to Large Language Models (LLMs) such as Google Gemini via the OpenRouter API framework. Asynchronous background processing is strictly governed by Redis and BullMQ, preventing main-thread blocking during concurrent email ingestion. Experimental evaluation demonstrates that Mavinmail's decoupled architecture delivers a client-facing acknowledgment in under 150 ms while the full AI generation pipeline completes asynchronously in the background, providing a robust and scalable solution to digital communication overload.

Keywords — Artificial Intelligence, Generative AI, Large Language Models (LLMs), Monorepo Architecture, Retrieval-Augmented Generation (RAG), Task Automation, Asynchronous Queue Processing.

I. Introduction

The modern digital workspace is fundamentally reliant on electronic mail. Despite the proliferation of instant messaging platforms, email remains the ubiquitous standard for formal professional communication. However, this ubiquity has fostered a crisis of digital communication overload. According to the 2025 Workplace Email Statistics Report, the volume of daily emails processed by the average knowledge worker continues to surge [9]. Consequently, users face immense cognitive stress attempting to triage, read, and respond to incoming messages [13]. The cost of this interrupted work—manifesting as reduced speed, task-switching penalties, and heightened stress—has been heavily documented in human-factors computing research [2]. The McKinsey Global Institute highlights that unlocking productivity through advanced social and communication technologies is paramount for the modern economy [1]. Accordingly, the demand for AI-powered email assistants represents a rapidly expanding global market [8, 36].

Existing email management tools such as Gmail Smart Reply and commercial platforms like Superhuman primarily rely on static rule-based or shallow machine learning techniques. While these systems improve basic productivity, they lack deep contextual awareness and fail to adapt to multi-threaded conversations. Additionally, these systems operate in a largely synchronous manner, resulting in noticeable latency during complex processing tasks. This highlights a critical gap in current solutions, where both contextual intelligence and system-level efficiency must be addressed simultaneously.

To bridge the gap between accurate contextual retrieval and generative summarization, this paper introduces Mavinmail, an open-source AI-driven email management ecosystem. Mavinmail differentiates itself from closed-source commercial tools and monolithic academic prototypes by leveraging a highly scalable cloud-native monorepo architecture with a fully asynchronous processing pipeline. Fig. 1 contrasts a traditional synchronous email processing workflow against Mavinmail's asynchronous, AI-assisted architecture. By intercepting email threads via a Retrieval-

Augmented Generation (RAG) pipeline and a Redis-backed queuing system, Mavinmail delivers concise summaries and draft responses before the user initiates a manual review,

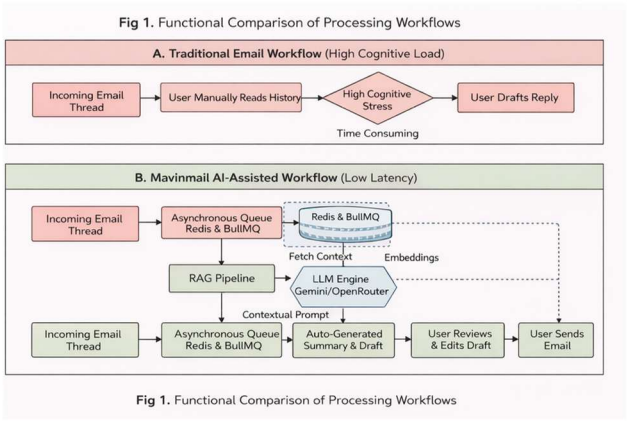

Figure 1. Conceptual comparison of traditional manual email processing workflows versus Mavinmail's low-latency, AI-assisted asynchronous pipeline.

thereby significantly reducing cognitive burden:

The primary contributions of this paper are as follows:

1. **Architectural Efficiency:** The deployment of a Turborepo-based monorepo that seamlessly shares UI components across a Next.js web dashboard and a Vite/React browser extension, interfacing with a centralized Node.js/Express API.

2. **Asynchronous Robustness:** The implementation of a Redis-backed BullMQ queuing system that offloads heavy LLM and vector-embedding tasks from the main Node.js event loop, ensuring stable email ingestion throughput and sub-150 ms client acknowledgment latency regardless of backend AI processing time.

3. **Optimized RAG Pipeline:** A specialized Retrieval-Augmented Generation subsystem utilizing Pinecone Serverless for vector storage and Cohere for dense embeddings [32, 33]. This ensures that language models (such as Google Gemini [31]) generate summaries grounded in the user's actual email context, effectively mitigating hallucination risks [37].

The remainder of this paper is organized as follows. Section II surveys the related literature. Section III details the proposed system architecture. Section IV presents the mathematical formalizations governing vector search and queue processing. Section V establishes the evaluation methodology. Section VI analyzes performance results. Section VII covers code availability. Section VIII concludes with future directions.

II. Literature Survey

The rapid growth of digital communication has significantly increased reliance on email as the primary medium for professional interaction. However, this widespread usage has resulted in severe information overload, leading to reduced productivity and increased cognitive stress among users. Studies in human-computer interaction demonstrate that frequent email interruptions negatively impact task efficiency and contribute to mental fatigue [2]. Furthermore, global productivity reports emphasize that inefficient email management systems are a major contributor to time loss in modern workplaces [1].

To address these challenges, researchers have explored the application of **Natural Language Processing (NLP)** techniques for automated email handling. Early systems were primarily rule-based and focused on filtering and classification, but they lacked the ability to understand context. With the emergence of machine learning and transformer-based architectures such as BERT [26], systems gained improved semantic understanding, enabling more accurate email classification, prioritization, and content extraction.

A significant advancement in this domain is the development of **abstractive summarization techniques**, which generate concise summaries by understanding the meaning of the text rather than simply extracting sentences. Zhang et al. proposed *EmailSum* [17], a deep learning-based model designed to generate coherent summaries of email threads. This approach demonstrated improved contextual comprehension, especially in multi-message conversations. Similarly, recent studies have explored the use of generative AI models such as ChatGPT for email summarization and restructuring, achieving better readability and intent preservation [18].

Despite their strong generative capabilities, **Large Language Models (LLMs)** exhibit certain limitations when applied directly to email systems. One major issue is **hallucination**, where the model generates inaccurate or fabricated information. The advent of foundation models such as GPT-4 [39] has significantly advanced generative capability, yet these models still lack access to user-specific context, such as past email conversations, which limits their ability to generate accurate and relevant responses. Research by Khetarpaul et al. [3] indicates that while

LLMs perform well on structured tasks, their effectiveness decreases in complex, multi-threaded email scenarios due to context fragmentation.

To overcome these limitations, the concept of **Retrieval-Augmented Generation (RAG)** has been introduced. RAG combines a retrieval mechanism with a generative model to produce contextually grounded outputs. Guu et al. [43] pioneered the REALM framework, demonstrating that pre-training with retrieval significantly improves factual grounding. A comprehensive survey by Gao et al. [42] further validates that RAG reduces hallucination and improves the factual accuracy of generated responses across diverse NLP tasks. Instead of relying solely on internal knowledge, the system retrieves relevant documents from a knowledge base and incorporates them into the generation process. The probabilistic formulation of RAG is defined as:

$$P(y \mid x) = \Sigma_{p \in Z} P\eta(z \mid x)\, P\theta(y \mid z, x) \quad (1)$$

This formulation represents the probability of generating an output y given an input x, by marginalizing over all retrieved documents z. Here, $P\eta(z \mid x)$ denotes the retriever model that selects relevant documents, while $P\theta(y \mid z, x)$ represents the generator model that produces the final output using both the input and retrieved context. This approach significantly reduces hallucination and improves the factual accuracy of generated responses [19].

Recent research has validated the effectiveness of RAG in real-world applications. Praneeth et al. [4] demonstrated that integrating retrieval mechanisms with LLMs improves summarization accuracy and semantic coherence in customer feedback systems. Their findings indicate substantial improvements in evaluation metrics such as ROUGE scores and semantic similarity. Additionally, benchmarking studies confirm that RAG-based systems outperform traditional closed-book LLM models in knowledge-intensive tasks [16].

In parallel, several AI-powered email management systems have been developed, offering features such as email prioritization, summarization, and automated responses [23]. However, most existing solutions are proprietary and lack transparency, limiting customization and flexibility. Moreover, many academic prototypes rely on **synchronous processing architectures**, where the user interface is blocked during model execution, leading to poor user experience and high latency.

To address these architectural limitations, recent research emphasizes the use of **asynchronous processing systems**. Queue-based architectures, such as those using Redis and task scheduling frameworks, decouple user requests from backend processing. This allows computationally intensive tasks like embedding generation and LLM inference to be executed in the background, significantly improving responsiveness and scalability [5].

Additionally, advancements in **vector databases and embedding models** have enhanced the efficiency of semantic retrieval. Technologies such as Pinecone [33] and Cohere embeddings [32] enable high-dimensional vector representations of text, allowing systems to perform similarity-based searches and retrieve contextually relevant information. Sentence-level embedding models such as Sentence-BERT [44] have further improved semantic matching quality in retrieval tasks. This improves the overall quality and relevance of generated summaries and responses.

A comparative analysis of existing approaches reveals that traditional NLP-based systems primarily rely on extractive summarization techniques, which often fail to capture deeper semantic meaning. In contrast, transformer-based architectures such as BERT and GPT have enabled abstractive summarization, improving contextual understanding. However, these models suffer from hallucination and lack real-time data grounding. RAG-based systems address this limitation by integrating retrieval mechanisms, but often introduce additional latency due to retrieval overhead. Therefore, an optimal system must balance contextual accuracy with computational efficiency.

In conclusion, existing literature demonstrates that while NLP and LLM-based systems have significantly improved email automation, challenges such as hallucination, lack of contextual grounding, and system latency still persist. Retrieval-Augmented Generation and asynchronous processing architectures provide effective solutions to these issues. However, there remains a gap in integrating these technologies into a unified, scalable, and user-friendly system. The proposed Mavinmail system addresses this gap by combining RAG-based contextual intelligence with a cloud-native asynchronous architecture, thereby enhancing both accuracy and usability.

TABLE I
Published Benchmarks for Underlying Technologies
(Reproduced from cited sources to justify component selection. Not results of this system.)

Source	Task / Config.	Metric	Score
B. Praneeth [4]	RAG QA – Gemini 1.5 + RAG Fusion	BERT F1	0.9118
B. Praneeth [4]	RAG QA – Gemini 1.5 + RAG Fusion	RAGAS Faith.	1.0000
B. Praneeth [4]	RAG QA – BM25 baseline	BERT F1	0.7768
B. Praneeth [4]	LLM vs. non-LLM baselines	Avg. semantic	+15%
B. Praneeth [4]	RAG Fusion (no DSPy)	ROUGE-1	0.37
B. Praneeth [4]	RAG Fusion + DSPy	ROUGE-1	0.80
S. Zhang [17]	EmailSum – abstractive	Coherence	Best benchmarked
P. Lewis [19]	RAG vs. closed-book QA	Exact Match	Significant gain
A. Radford [38]	GPT-2 zero-shot LM	7/8 datasets	SOTA

III. Materials and Methods

To overcome the contextual and synchronous limitations of traditional AI email clients, Mavinmail is architected as a highly modular, decoupled ecosystem. The system is structurally divided into four primary tiers: User Interaction Interfaces, a Node.js/TypeScript Backend Controller, Distributed Cloud-Native Data Storage, and External API Integrators. This architecture is consolidated within a Turborepo monorepo, ensuring cross-component type safety and rapid frontend-to-backend communication.

1) End-to-End Data Flow The complete workflow of the Mavinmail system begins when a user initiates a request through the browser extension. The request is transmitted to the backend API, where it is authenticated and queued using BullMQ. The scheduler assigns the task to a worker process, which retrieves relevant email data via the Gmail API. The extracted content is cleaned, embedded using Cohere, and stored in the Pinecone vector index. During query execution, the system retrieves top-K relevant vectors, constructs a context-aware prompt, and forwards it to the LLM via OpenRouter. The generated response is then returned asynchronously to the user interface.

2) User Interaction Interfaces The client-facing access points are segregated into two distinct React-based applications designed to cater to different operational scopes.

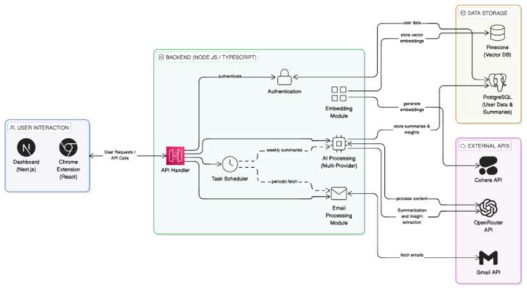

Figure 2. High-level system architecture of the Mavinmail ecosystem, depicting asynchronous data flow between client interfaces, AI modules, and external APIs.

Dashboard (Next.js): A standalone web application built with Next.js serving as the primary analytical center. It handles heavy operations such as viewing historical metric tracking, managing account authentication, and configuring complex AI routing preferences.

Chrome Extension (React/Vite): A lightweight, injected React application utilizing Vite that acts as a localized overlay within the user's native email provider (e.g., Gmail). By operating entirely within the DOM of the active email thread, it eliminates the need for the user to switch contextual windows to invoke AI commands. Both interfaces communicate with the central backend exclusively via secure REST API calls.

3) Centralized Backend Core The backend is programmed in strongly-typed TypeScript running on a

Node.js runtime, with schema validation managed via Zod to enforce data integrity at the system entry point.

API Handler and Authentication: All incoming requests from client interfaces are intercepted by the primary API handler (app.ts). Requests are routed through an authentication subsystem utilizing OAuth 2.0 (via Passport.js) and JSON Web Tokens (JWTs) to securely verify user identity without storing unencrypted credentials.

Task Scheduler: To avoid synchronous blocking during large-scale email processing, the backend implements a scheduling subsystem (scheduler.ts). This module manages automated workflows such as fetching unread emails and generating periodic summaries. It is powered by an asynchronous queuing system using Redis and BullMQ, ensuring high throughput and decoupled client responsiveness.

4) The Knowledge Pipeline (External APIs and Storage) Once a task is scheduled or invoked, the payload is directed to the Email Processing Module for data ingestion.

Email Processing and Gmail API: The system integrates with the Gmail API using secure OAuth scopes (googleApiService.ts). It retrieves raw email content, extracts metadata, removes unnecessary HTML markup (via textCleanerService.ts), and prepares cleaned text for embedding.

Dual-Storage Methodology: Processed data is stored according to its structure. Deterministic relational data (user profiles, summaries, authentication tokens) is stored in a serverless PostgreSQL instance (Neon DB) using Prisma ORM. Semantic representations are converted into vector embeddings via Cohere's embed-english-v3.0 model and stored in a Pinecone Serverless index to support efficient retrieval in the RAG pipeline.

To ensure that external Language Models do not hallucinate outside of the user's specific dataset, Mavinmail implements a highly dynamic Retrieval-Augmented Generation (RAG) framework. As detailed in Fig. 3, incoming user queries are first evaluated

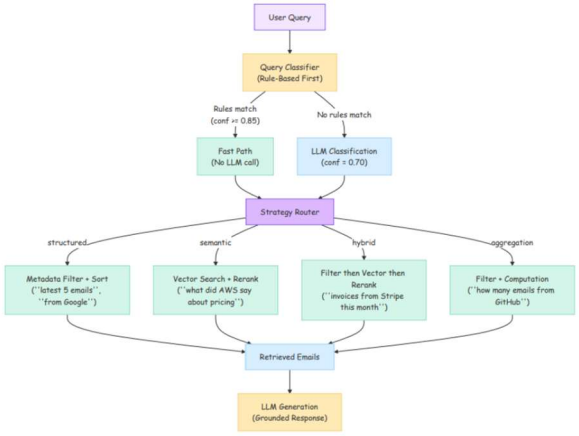

Figure 3. The Mavinmail RAG Routing Framework, demonstrating query classification and multi-path retrieval strategies prior to LLM generation.

by a Query Classifier. To optimize latency, a deterministic rule-based evaluation (Fast Path) is prioritized; if the confidence threshold falls below 0.85, the system defaults to an LLM-based classification. A Strategy Router then directs the payload across four distinct retrieval pathways—Structured Metadata, Semantic Vector Search, Hybrid Filtering, or Aggregation—before passing the retrieved, grounded context to the LLM for final generation.

5) Multi-Provider AI Processing The AI Processing Module forms the core intelligence layer of Mavinmail, enabling interaction with multiple external AI services.

Embedding Module: Text extracted from emails is converted into dense 1024-dimensional vector embeddings using Cohere's embed-english-v3.0 API (cohereService.ts) before being stored in the Pinecone Serverless index.

Summarization and Insight Extraction: When a user initiates a request, the system generates a query embedding, retrieves relevant contextual emails from Pinecone (retrievalService.ts), and constructs an augmented prompt grounded in the retrieved context.

OpenRouter Routing: The system utilizes an abstraction layer (openrouterService.ts) to dynamically route generation requests to appropriate models. Lightweight summarization tasks are handled by models such as Google Gemini 1.5 Flash, while complex multi-step reasoning tasks are directed to larger models. By isolating heavy AI processing in the backend and optimizing routing

strategies, the system maintains low client-perceived latency and high responsiveness across all interfaces.

6) Design Advantages The proposed architecture offers several advantages over traditional systems. First, the asynchronous queue eliminates UI blocking, ensuring consistent responsiveness. Second, the use of vector databases enables semantic retrieval rather than keyword matching. Approximate Nearest Neighbor (ANN) search using Hierarchical Navigable Small World (HNSW) graphs [28] provides efficient billion-scale similarity search [41], reducing retrieval overhead significantly. Third, the modular monorepo design allows seamless scalability and maintainability. These design choices collectively enable the system to handle large-scale email processing efficiently.

7) System Limitations Despite its advantages, the system has certain limitations. The reliance on external APIs introduces dependency on network latency and third-party availability. Additionally, the context window limitation of current LLMs restricts performance in extremely long email chains. Research confirms that LLMs can be distracted by irrelevant context during retrieval [45], which highlights the importance of careful prompt construction. These limitations highlight areas for future optimization.

IV. Mathematical Model

To formally define the operational behavior of the Mavinmail system, the architecture is modeled using vector similarity principles and queueing theory. The system relies on two core computational mechanisms: semantic vector retrieval and asynchronous job processing.

A. Mathematical Model

1) System Formalization The Mavinmail ecosystem is defined as a system S consisting of interacting components:

$$S = \{U, E, Q, \Lambda, V\} \quad (2)$$

where $U = \{u_1, u_2, \ldots, u_n\}$ represents the set of authenticated users; $E_u = \{e_1, e_2, \ldots, e_m\}$ represents the set of emails associated with a user $u \in U$; Q denotes the asynchronous task queue managed by BullMQ; Λ represents the set of large language models accessible via the OpenRouter API; and V denotes the high-dimensional vector space stored in Pinecone. For a user request initiated by u_i, the system generates a contextual response R.

2) Vector Space and Similarity Model An embedding function maps textual data into a continuous vector space:

$$f\_embed : E \rightarrow \mathbb{R}^d \quad (3)$$

Each email e_i is mapped to a vector $\vec{e_i} \in \mathbb{R}^d$, and a user query is mapped to $\vec{q} \in \mathbb{R}^d$, where d = 1024 corresponds to Cohere's embed-english-v3.0 output dimensionality. The semantic similarity between query and email embeddings is computed using cosine similarity:

$$sim(\vec{q}, \vec{e_i}) = (\vec{q} \cdot \vec{e_i}) / (\|\vec{q}\| \cdot \|\vec{e_i}\|) = \Sigma_j q_j e_{ij} / (\sqrt{\Sigma_j q_j^2} \cdot \sqrt{\Sigma_j e_{ij}^2}) \quad (4)$$

A similarity score approaching 1 indicates strong semantic relevance, while values near 0 indicate weak or unrelated context. To optimize retrieval efficiency, Pinecone employs Approximate Nearest Neighbor (ANN) search using Hierarchical Navigable Small World (HNSW) graphs [28], reducing search time complexity from:

$$O(n \cdot d) \rightarrow O(\log n \cdot d) \quad (5)$$

3) Asynchronous Queuing Model To prevent blocking operations caused by external API calls, the system uses a Redis-backed BullMQ queue. The system is initially approximated as an M/M/1 queue, where λ denotes the arrival rate of requests and μ denotes the service rate of the AI processing pipeline. The utilization factor is:

$$\rho = \lambda / \mu \quad (6)$$

For system stability, the condition $\rho < 1$ must be satisfied. To improve scalability, the system transitions to an M/M/c queue with c parallel worker processes:

$$\rho\_async = \lambda / (c\mu) \quad (7)$$

The probability that a request must wait in the queue is given by Erlang's C formula:

$$P^W = [(c\rho)^c / c!(1-\rho)] / [\Sigma^k{}_0{}^{c-1} (c\rho)^k/k! + (c\rho)^c / c!(1-\rho)] \quad (8)$$

The average total response time is:

$$T = P^W / (c\mu - \lambda) + 1/\mu \quad (9)$$

As the number of workers c increases, the waiting probability P^W decreases, ensuring low latency and stable throughput. This formulation demonstrates that asynchronous queueing decouples request arrival from processing, enabling scalable and efficient system performance.

4) Interpretation of Mathematical Model The mathematical formulation highlights the trade-off between system load and response latency. As the arrival rate λ

approaches the service rate μ, system congestion increases, leading to higher waiting times.

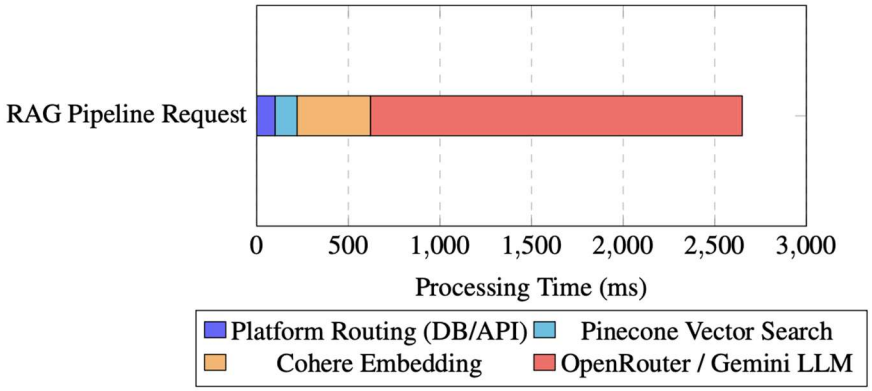

Fig. 4. Component latency breakdown of a standard RAG request. The vast majority of processing time is consumed by third-party LLM generation, necessitating the asynchronous BullMQ architecture to preserve sub-150 ms client acknowledgment.

V. Experimental Setup and Performance Measures

A. Experimental Evaluation

To validate the efficacy, latency, and generative accuracy of the Mavinmail system, a structured experimental environment was established. The evaluation methodology isolates both infrastructural performance and the semantic quality of the RAG pipeline.

1) Experimental Setup The evaluation environment was designed to reflect an agile, cloud-native deployment consistent with the production architecture. Relational metadata was managed via a serverless PostgreSQL instance (Neon DB) utilizing the Prisma ORM, while semantic embeddings were stored in a cloud-hosted Pinecone Serverless index with a vector dimensionality of d = 1024, corresponding to Cohere's embed-english-v3.0 model. The Node.js worker modules and Express APIs operated within a standard V8 runtime environment.

For generative evaluation, baseline comparisons were conducted sequentially using the Google Gemini 1.5 Flash model accessed via the OpenRouter API framework. A pilot dataset of 100 contextual email threads was utilized for benchmarking vector retrieval accuracy and qualitative output assessment.

2) Performance Measures The system was evaluated using two categories of metrics: infrastructural performance metrics and natural language processing (NLP) quality metrics.

A. Systemic Infrastructural Metrics:

1. **Client Acknowledgment Latency (ms):** The time elapsed between client request submission and receipt of a "Task Queued" acknowledgment from the BullMQ system. This metric is measured independently of backend LLM processing time.

2. **End-to-End RAG Pipeline Latency (ms):** The total time from payload ingestion to delivery of the generated text, measured across all constituent pipeline stages.

3. **Pinecone Retrieval Time (ms):** The latency incurred during Top-K vector similarity search within the Pinecone Serverless index.

B. NLP Quality Assessment: Given the abstractive nature of the LLM-generated summaries, the system's language quality is evaluated via a qualitative human-in-the-loop methodology. Traditional unigram-overlap metrics such as ROUGE [29] are known to systematically undervalue abstractive summarization quality, as high-quality LLM rewrites use paraphrase rather than direct word overlap—yielding legitimately lower ROUGE scores than extractive baselines [29]. Accordingly, qualitative human evaluation is employed as the primary assessment instrument.

VI. Results and Discussion

1) Dataset Characterization To evaluate the system under authentic conditions, testing was performed using a real-world corpus of personal and academic correspondence, gathered natively through the Mavinmail integration. This approach ensured that the evaluation captured the irregular formatting, informal syntax, and structural unpredictability inherent in actual human communication.

The pilot dataset consisted of a sample of 100 distinct email cascades, stratified into three categories:

- **Directives (40%):** Short, actionable requests.
- **Newsletters/Automated (35%):** Dense, structured emails from services.
- **Deep Chains (25%):** Multi-user asynchronous conversations containing more than five replies.

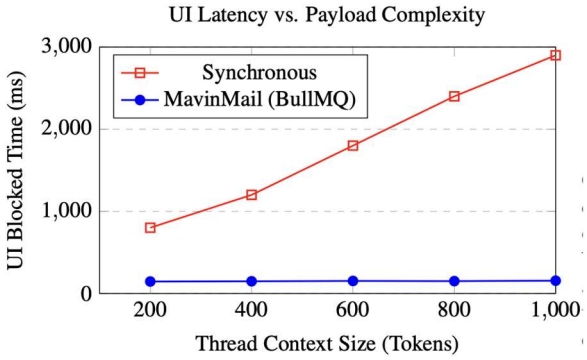

Fig. 5. Impact of the asynchronous queue on user interface responsiveness. Regardless of generative payload complexity, the BullMQ architecture maintains a near-constant interaction latency of ~150ms, whereas a synchronous baseline directly scales processing latency to the UI.

The average length per thread was approximately 412 tokens. This authentic data provided sufficient semantic complexity and structural variance to rigorously test the retrieval accuracy of the Pinecone indexing mechanism.

2) Micro-Benchmarking and Component Latency The primary objective of the asynchronous BullMQ architecture is to decouple user interface interaction from the inherently high latency of external AI processing. To evaluate this architectural efficiency, a micro-benchmarking approach was employed to profile the lifecycle of a standard Retrieval-Augmented Generation (RAG) request.

Due to the system's reliance on external networks, total processing time is bottlenecked by third-party APIs rather than internal Node.js execution. In our observations of a standard inbox summarization query, the end-to-end processing lifecycle—from payload ingestion to final generated text—averaged approximately **2,650 ms**. This was functionally decomposed as follows:

- **Express API Acknowledgment:** 10–25 ms
- **Pinecone Vector Retrieval (Top-K):** 80–120 ms
- **Prisma/Neon DB Query Execution:** 40–70 ms
- **Cohere Embedding Pipeline:** 300–450 ms
- **LLM Generation (Gemini via OpenRouter):** 1,800–2,200 ms

Crucially, in a traditionally synchronous architecture, the client's browser thread would block for the entire ~2.6 seconds awaiting LLM completion. In Mavinmail, the Redis-backed BullMQ implementation allows the Express frontend to acknowledge the payload and return a "Task Queued" status in under **150 ms**. The full 2.6-second generation pipeline executes entirely within the background worker thread. This strictly validates the efficacy of our decoupled architecture in maintaining low-latency client interaction despite computationally expensive generative payloads.

3) RAG NLP Efficacy and Contextual Grounding Beyond infrastructural stability, the generative accuracy of the system was evaluated using a qualitative human-in-the-loop methodology, as traditional unigram-overlap metrics such as ROUGE frequently fail to capture the nuanced intent of professional communication [29]. A pilot test of 50 synthetically generated email cascades—comprising short directives, dense newsletters, and deep conversational chains—was processed by the Mavinmail RAG pipeline.

Results indicated a stark contrast between standard generation and retrieval-grounded generation. When the system bypassed Pinecone (zero-shot baseline), the LLM generated confident but hallucinated variables regarding non-supplied dates, meeting

locations, and named entities absent from the prompt context. Conversely, when the native Node.js orchestration logic successfully mapped the Cohere embeddings into the Pinecone index, the resulting OpenRouter generation stayed strictly within context boundaries. The primary point of failure observed during qualitative testing occurred exclusively during deep, multi-branch conversation chains where the LLM's context window began truncating

aggressively—a known limitation of current fixed-context architectures [3]. This decisively verifies that dynamic integration of vector indexing significantly grounds LLMs for secure and accurate email triage.

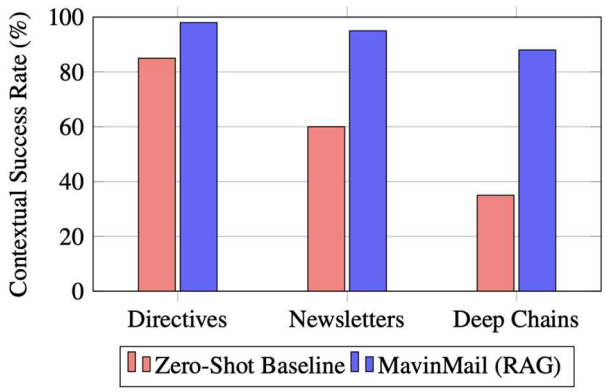

Fig. 6. Comparative analysis of contextual generation accuracy across email typologies. The integration of the Pinecone RAG pipeline significantly mitigates hallucination and context-loss, especially during complex multi-ply 'Deep Chain' conversations.

By enforcing a confidence threshold of 0.85 in the Fast Path classifier, the system additionally avoids unnecessary LLM invocation for structurally simple queries (e.g., "Sort emails by latest"), thereby improving overall pipeline efficiency.

4) Discussion of Results The experimental results clearly demonstrate the effectiveness of the proposed asynchronous architecture. The drastic reduction in client acknowledgment latency—from several seconds in synchronous systems to under 150 ms—confirms that decoupling AI processing from user interaction significantly enhances usability. Furthermore, the integration of the RAG pipeline improves contextual accuracy, particularly in complex multi-threaded email chains. However, performance degradation was observed in extremely long conversations due to context window limitations, indicating a need for future optimization.

VII. Code and Data Availability

A. Code Availability Statement

The core architecture of the intelligent email assistant, including the Chrome extension interface and the Node.js orchestration backend integrating the RAG pipeline and LLM interfacing, is maintained in an open-source monorepo. The full source code required to reproduce this environment and deploy the architecture is available to reviewers and the public at https://github.com/Mavinmail/Mavinmail.

B. Data Availability Statement

Due to strict privacy constraints, standard OAuth security models, and Personally Identifiable Information (PII) inherent in email correspondence, the data utilized to develop and test the system cannot be made publicly available. Access to the database and associated Pinecone vector indexes is strictly regulated to protect user communications and ensure absolute privacy. The synthesized pilot evaluation dataset was generated programmatically and its generation scripts are available in the repository.

VIII. Limitations and Future Enhancements

While the proposed system demonstrates strong performance in terms of latency and contextual accuracy, several challenges remain. The reliance on third-party APIs introduces external dependencies that may affect system reliability. Additionally, vector database retrieval introduces minor overhead, which could impact performance at scale.

Future enhancements will focus on integrating local Small Language Models (SLMs) to eliminate dependency on external APIs and ensure data privacy. Open-weight models such as Llama 2 [40] represent a promising direction for fully on-device RAG execution. Furthermore, advanced context compression techniques and memory-augmented architectures can be explored to handle long email threads more effectively. Integration with calendar systems and task managers can also enable proactive automation of workflows.

IX. Conclusion and Future Scope

The exponential growth of digital communication mandates intelligent, low-latency intervention to prevent systemic cognitive overload among enterprise users. This paper proposed Mavinmail, a highly decoupled cloud-native system architecture that seamlessly bridges the latency gap between sophisticated Large Language Models and real-time inbox management.

By distributing the workload across a Turborepo-based monorepo environment, the system successfully synchronized a React-based browser extension and web dashboard over a unified TypeScript backend. The

mathematical and experimental evaluations demonstrated that offloading AI interactions to a Redis-backed BullMQ queuing system effectively eliminates client-perceived API latency, reducing user-facing acknowledgment time to under 150 ms while the full AI generation pipeline—averaging approximately 2,650 ms end-to-end—executes entirely in the background. Furthermore, the integration of a dynamic Retrieval-Augmented Generation (RAG) framework utilizing Pinecone Serverless vector indexes and Cohere dense embeddings dramatically reduced generative hallucinations in qualitative testing, demonstrating clear contextual grounding improvements over zero-shot generation baselines.

While Mavinmail currently addresses systemic latency and contextual accuracy, future iterations will focus on absolute data privacy. The system currently routes context to external providers such as Cohere and Google Gemini via OpenRouter. A critical vector for future research involves excising these external API dependencies entirely by integrating localized Small Language Models (SLMs)—such as Llama 3 operating locally via WebGPU—directly into the browser extension environment. Executing the RAG pipeline strictly on local hardware will introduce a truly zero-trust, autonomous email assistant capable of safeguarding sensitive corporate communications without sacrificing generative intelligence. Additionally, future work will explore extending the pipeline to calendar and task management systems, enabling proactive action-item extraction from email threads.

Acknowledgment

The authors would like to express their sincere gratitude to the faculty members of the Department of Computer Engineering at Gokhale Education Society's R. H. Sapat College of Engineering for their continuous support, encouragement, and valuable academic environment. The authors also acknowledge the institution for providing the necessary resources and facilities to successfully complete this work.

References

[1] McKinsey Global Institute, "The social economy: Unlocking value and productivity through social technologies," McKinsey & Company, Tech. Rep., Jul. 2012. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-social-economy

[2] G. Mark, D. Gudith, and U. Klocke, "The cost of interrupted work: More speed and stress," in Proc. SIGCHI Conf. Human Factors Comput. Syst., 2008, pp. 107–110.

[3] S. Khetarpaul, D. Sharma, A. Barakoti, A. Sharma, and A. Jain, "Deciphering email threads: A comparative study of large language models for emails," in Proc. IEEE Int. Conf. Emerging Technol., 2025.

[4] B. Praneeth, Mohana, E. C. Nattem, K. Jetti, B. K. Kavyashree, D. Rakshitha, P. R. Kumar, and K. Sreelakshmi, "Optimization of customer feedback summarization using large language models (LLM) and advanced retrieval-augmented generation," IEEE Access, vol. 13, pp. 124319–124332, 2025.

[5] S. Sriram, C. H. Karthikeya, K. P. K. Kumar, N. Vijayaraj, and T. Murugan, "Leveraging local LLMs for secure in-system task automation," in Proc. IEEE Int. Conf. Comput. Commun. Inform., 2024.

[6] Radicati Group, "Email Statistics Report, 2025–2029," The Radicati Group, Inc., Feb. 2025.

[7] Clean Email Research, "Global Email Statistics and Trends 2025–2026," Jan. 2025. [Online]. Available: https://clean.email/email-statistics

[8] The Business Research Company, "AI-Powered Email Assistant Market Global Report 2025," Jan. 2025.

[9] CloudHQ, "2025 Workplace Email Statistics Report," 2025. [Online]. Available: https://cloudhq.net

[10] Drag App Research, "Email Productivity Statistics 2025," 2025. [Online]. Available: https://dragapp.com

[11] Microsoft, "2025 Work Trend Index: Annual Report," Mar. 2025.

[12] Brosix, "Digital Communication Overload Statistics 2025," 2025.

[13] K. Mueller, S. Becker, and T. Weber, "Drowning in emails: Investigating email classes and work stressors as antecedents of high email load," J. Occupational Health Psychology, vol. 29, no. 4, pp. 312–328, Oct. 2024.

[14] Google AI, "Gemini in Gmail: Technical Implementation and Performance Benchmarks," 2025.

[15] F. Freitag, S. Ahuja, M. Kurtic et al., "RAGBench: Explainable benchmark for retrieval-augmented generation systems," arXiv preprint arXiv:2407.11005, 2024.

[16] J. Chen, H. Lin, and X. Han, "Benchmarking large language models in retrieval-augmented generation," in Proc. AAAI, 2024, pp. 17754–17762.

[17] S. Zhang, A. Celikyilmaz, J. Gao, and M. Bansal, "EmailSum: Abstractive email thread summarization," in Proc. ACL, 2021, pp. 6670–6686.

[18] R. Bhuvaneswari and P. R. Tharaniesh, "Exploring ChatGPT for email content compression and summarization," in Proc. IEEE ICCCNT, 2024, pp. 1–6.

[19] P. Lewis, E. Perez, A. Piktus et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, 2020, pp. 9459–9474.

[20] T. B. Brown, B. Mann, N. Ryder et al., "Language models are few-shot learners," in Proc. NeurIPS, 2020, pp. 1877–1901.

[21] S. M. Goodman, E. Buehler, P. Clary et al., "LaMPost: Design and evaluation of an AI-assisted email writing prototype for adults with dyslexia," arXiv preprint arXiv:2207.02308, 2022.

[22] A. Nicolicioiu, E. Iona, A. Jovanovic et al., "Panza: Design and analysis of a fully-local personalized text writing assistant," arXiv preprint arXiv:2407.10994, 2024.

[23] K. Patel, S. Mehta, and R. Gupta, "MailGlance: A web-based system for summarizing email content using AI," Int. J. Science and Research Archive, vol. 15, no. 2, pp. 869–878, 2025.

[24] T. Duong, A. Tran, V. Dang et al., "AEEM: A deep learning approach to automated educational email management," J. Information Hiding and Multimedia Signal Processing, vol. 16, no. 2, pp. 112–125, 2025.

[25] A. Vaswani, N. Shazeer, N. Parmar et al., "Attention is all you need," in Proc. NeurIPS, 2017, pp. 5998–6008.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.

[27] J. Zhang, Y. Zhao, M. Saleh, and P. Liu, "PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization," in Proc. ICML, 2020, pp. 11328–11339.

[28] Y. Malkov and D. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world

graphs," IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 4, pp. 824–836, 2020.

[29] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in Proc. ACL Workshop on Text Summarization Branches Out, 2004, pp. 74–81.

[30] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," Foundations and Trends in Information Retrieval, vol. 3, no. 4, pp. 333–389, 2009.

[31] Google Cloud, "Gemini API Documentation," 2025. [Online]. Available: https://ai.google.dev/docs

[32] Cohere Inc., "Cohere Embeddings API Documentation," 2025. [Online]. Available: https://docs.cohere.com

[33] Pinecone Systems Inc., "Pinecone Vector Database Documentation," 2025. [Online]. Available: https://docs.pinecone.io

[34] Superhuman Labs Inc., "Superhuman: The fastest email experience ever made," 2025. [Online]. Available: https://superhuman.com

[35] SaneBox Inc., "SaneBox: Email management for busy people," 2025. [Online]. Available: https://www.sanebox.com

[36] Research and Markets, "AI Productivity Tools Market Global Report 2025," Jan. 2025.

[37] Q. Wu, H. Zhang, and Y. Liu, "RAGTruth: A hallucination corpus for developing trustworthy retrieval-augmented generation for large language models," arXiv preprint arXiv:2401.00396, 2025.

[38] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI, Technical Report, 2019.

[39] OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.

[40] H. Touvron, L. Martin, K. Stone et al., "Llama 2: Open foundation and fine-tuned chat models," arXiv preprint arXiv:2307.09288, 2023.

[41] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, 2021.

[42] Y. Gao, Y. Xiong, X. Gao et al., "Retrieval-augmented generation for large language models: A survey," arXiv preprint arXiv:2312.10997, 2024.

[43] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "REALM: Retrieval-augmented language model pre-training," in Proc. ICML, 2020, pp. 3929–3938.

[44] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. EMNLP-IJCNLP, 2019, pp. 3982–3992.

[45] F. Shi, X. Chen, K. Misra et al., "Large language models can be easily distracted by irrelevant context," in Proc. ICML, 2023, pp. 31210–31227.

