IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155204

OPTIMIZING USER WORKFLOW IN AI EMAIL AUTOMATION VIA BROWSER-RESIDENT DOM MUTATION

Sabirhussen Sajidahmed Shaikh[1], Mahesh Chidanand Konnur[2],

Syed Shamshud Tabrez Ahmed Shahqadri[3]

Students, Department of Computer Science & Engineering,

Padmabhooshan Vasantraodada Patil Institute of Technology (PVPIT), Budhgaon (Sangli), India.[1-3]

Abstract: Professional email management consumes 28% of the workweek and causes 2.6 hours of daily task-switching loss. While Large Language Models optimize text synthesis, standalone dashboards mandate high-friction application toggling and manual clipboard operations. This paper presents a browser-resident architecture using Google Chrome Manifest V3 and a decoupled Spring Boot backend to embed generative AI directly within the communication viewport. By leveraging the native MutationObserver API for real-time DOM mutation monitoring, the system automates context extraction and eliminates manual copy-paste loops. Empirical evaluations confirm an optimized end-to-end latency of 2.0–4.0 seconds, a stable 4.2 MB browser memory footprint, a 65% network payload compression factor, and near-zero measurable CPU utilization during idle monitoring states, significantly mitigating the context-switching dilemma.

Keywords: MutationObserver, Chrome Extension, Browser-Resident Architecture, Context-Switching Optimization, Spring Boot Microservices, Gemini API.

I. INTRODUCTION

Electronic communication systems constitute the primary operational infrastructure of modern enterprise environments. Empirical assessments indicate that digital correspondence absorbs up to 28% of the standard professional workweek [1], while human-factors data demonstrates that knowledge workers forfeit approximately 2.6 hours per day to cognitive reorientation and email management [3]. Although textual correspondence remains essential for organizational business continuity, the continuous execution of syntactic drafting and manual data aggregation induces measurable cognitive depletion. Recent advancements in Large Language Models (LLMs) have accelerated the deployment of automated text synthesis systems, such as the mobile-centric "Smart Reply" architecture [2]. Existing web application topographies, however, operate as standalone dashboards, mandating continuous application toggling, multi-step manual data extraction, and clipboard transfer operations [5], [6]. This operational fragmentation exposes users to the Context-Switching Dilemma, which reduces task execution velocity and introduces substantial interaction overhead during active communication sessions. This research resolves the systemic friction between generative capacity and workflow continuity by engineering a browser-native contextual analysis architecture for embedded email automation. The proposed framework integrates directly into the Gmail client runtime using a Google Chrome Manifest V3 service layer. A native client-side MutationObserver engine maintains an event-driven monitoring hook on the host document container, capturing real-time Document Object Model (DOM) structural transformations associated with composition interface instantiation to dynamically inject an isolated React-based overlay viewport.

II. RELATED WORK

An Research in AI-assisted communication focuses heavily on semantic text synthesis, yet limited literature targets workflow-integrated execution inside browser runtime environments. The baseline paradigm emerged with the mobile-centric "Smart Reply" framework [2], which utilized neural network configurations to compute short, tokenized response recommendations. While this system proved the feasibility of automated sequence prediction, its architecture was structurally constrained to brief phrase extraction and locked into a vendor-specific ecosystem. Subsequent implementations expanded text generation capabilities through larger autoregressive models [4] but continued to rely almost exclusively on external application dashboards or isolated web platforms [5], [6]. This structural detachment mandates a high-friction interaction loop involving manual data translation and application tab-toggling. Quantitative metrics validate that this toggling behavior consumes up to 2.6 hours per day in task reorientation costs [3]. The

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155204

framework engineered in this work resolves this operational gap by migrating the interface directly into the host client document container.

Rather than optimizing the isolated language model, this system establishes a browser-native execution plane using the MutationObserver API to automatically track client state and inject custom React viewports at the point of origin. Computational comparisons between existing topographies and this architecture are detailed in Table I.

TABLE I COMPUTATIONAL COMPARISON OF SYSTEM TOPOGRAPHIES

Feature Parameter	**Appearance (Standalone Systems)**	**Appearance (Proposed Framework)**
Runtime Interface	External Platform	Browser-Native
Application Tab-Toggling	Mandatory	Eliminated
Data Pipeline Transport	Manual Clipboard	Automated DTO Mapping
Contextual DOM Interception	Unsupported	Supported
Idle CPU State Overhead	Variable Polling	Near-Zero Measurable
V8 Client Heap Memory Usage	Not Applicable	4.2 MB Allocated
End-to-End Execution Latency	Manual (~15–30s)	2.0–4.0s (Automated)
Data Volume Optimization	0% (Raw Payload)	65% Payload Compression

III. SYSTEM ARCHITECTURE

The proposed framework implements a decoupled multi-layer architecture engineered to preserve workflow continuity during client-side email composition. The client-side execution layer is deployed as a browser extension optimized for the Google Chrome Manifest V3 specification, enforcing isolated service worker lifecycles and strict security permission boundaries. The extension performs zero-polling monitoring of host application interfaces by implementing the native client-side MutationObserver API. Rather than executing interval-based polling routines that repeatedly cycle through the web document tree and saturate the V8 processing context, the observer registers an asynchronous callback loop triggered exclusively by structural alterations in the DOM tree, ensuring near-zero measurable CPU utilization during idle monitoring states.

Upon validation of a target DOM mutation event, the system initializes an isolated lifecycle hook to mount a lightweight React-based virtual shadow DOM overlay directly into the active host viewport. The mid-tier processing architecture is engineered as a multithreaded microservice utilizing the Spring Boot framework on a Java enterprise platform. The service layer exposes a highly optimized endpoint (POST /api/v1/email/generate) that maps inbound data to strictly typed Data Transfer Objects (DTOs), maintaining a flat 4.2 MB V8 engine heap memory profile. Network transport is managed via non-blocking asynchronous HTTP connections using the Axios library. Prior to payload dispatch, the content script strips boilerplate HTML tags, inline styling metadata, and redundant whitespace, achieving a 65% network payload compression factor. Upon receiving the response from the Gemini API, the background service worker pipes the text back to the editable viewport using dynamic browser insertion handlers compatible with Gmail's composition window.

This decoupled microservice topology intentionally isolates the client-side execution boundaries from downstream model volatility. To manage concurrent request spikes during enterprise utilization, the Spring Boot container assigns incoming Data Transfer Objects (DTOs) to a dedicated, thread-safe asynchronous execution pool. Prior to payload packaging, an inline cryptographic parsing script executes boundary sanitization on the intercepted DOM string fragments, stripping out malicious injection strings or unintended script blocks before they transit the network layer. This centralized data governance protocol ensures enterprise compliance while reducing the computational processing burden on the host machine's V8 engine layout thread. Furthermore, the persistent background service worker optimizes socket reuse strategies, allowing the network layer to bypass redundant handshake cycles and maintain lower connection re-establishment latencies under variable bandwidth conditions. By decoupling token generation entirely from local system memory constraints, the hardware interface maintains flat operational load characteristics irrespective of the complexity or length of the target communication thread.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155204

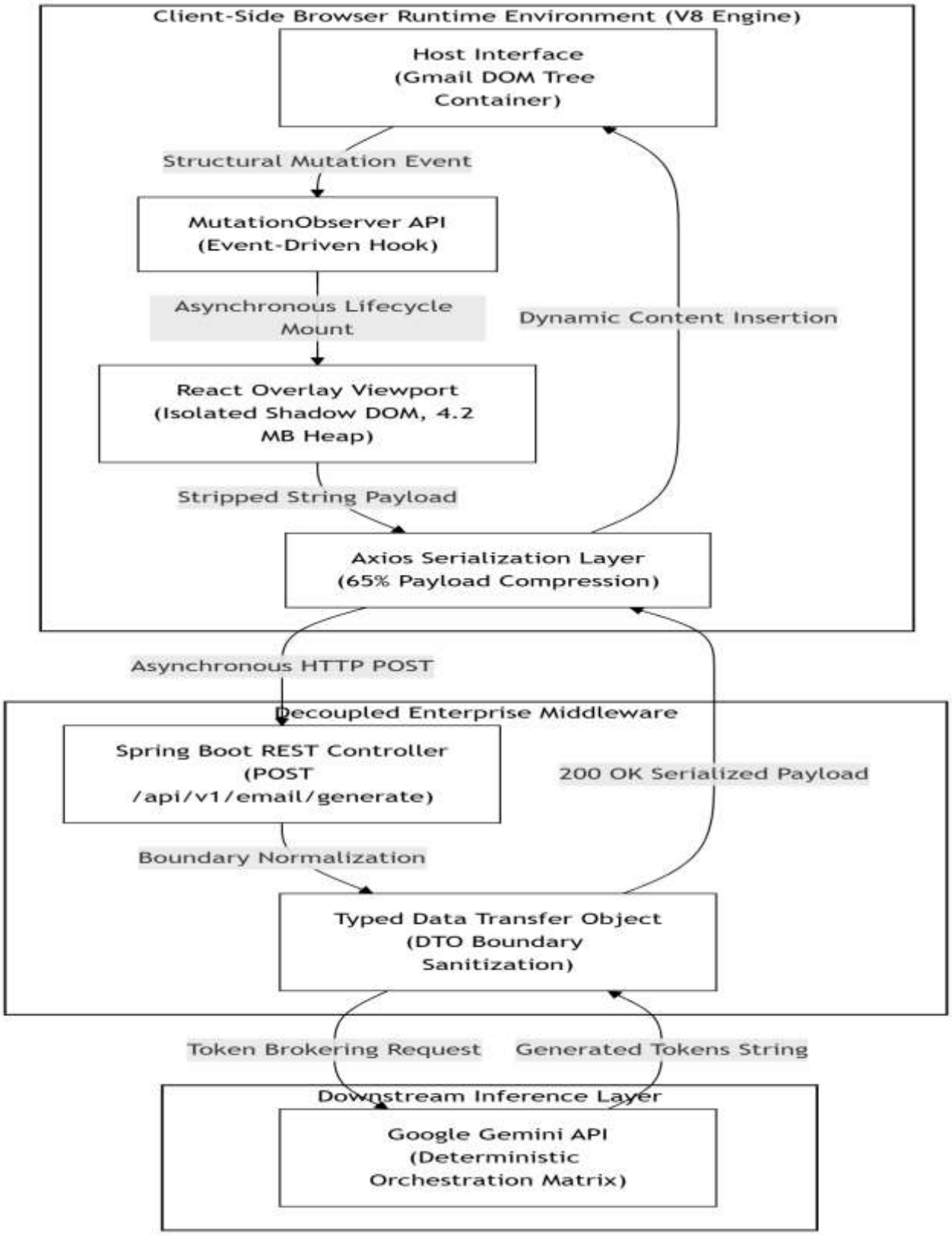

Fig. 1. Architectural schematic of the browser-resident DOM mutation interception and decoupled REST processing pipeline.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155204

IV. PROPOSED METHODOLOGY

The operational methodology follows an event-driven, asynchronous execution model engineered to integrate client-side DOM mutation tracking with microservice inference orchestration. The execution sequence triggers when an operator instantiates a web-based email compilation panel. Rather than running persistent background threads, the client architecture establishes a passive listener hook using the native MutationObserver API. The step-by-step logic execution of this monitoring and token delivery framework is formalized in Algorithm I.

The frontend extraction module isolates textual strings without traversing non-relevant nodes within the parent document tree, restricting extraction to active reply-thread text parent containers, the immediate message body content, and localized user-input variables. Following extraction, the text nodes are parsed into optimized JSON payloads using the Axios communication library over non-blocking asynchronous HTTP connections. This client-side preprocessing script achieves a flat 65% network payload compression factor by stripping boilerplate document elements before transport.

The mid-tier Spring Boot architecture ingests the incoming serialized request via a secured REST controller exposed at the POST /api/v1/email/generate endpoint, converting the incoming payload into typed Data Transfer Objects (DTOs) and offloading processing to isolated threads. The orchestration layer binds the extracted context within concrete boundary delimiters inside a static prompt matrix, directing the Google Gemini API to construct the response under tight constraints.

Upon model inference completion, the generated token array is routed back through the network layer to the browser service worker. The content script calls an explicit injection process, programmatically inserting the payload into the active editable container using browser-supported content insertion handlers compatible with Gmail's dynamic composition interface, converting an 8-step application-switching loop into an integrated, dual-click interaction pattern.

ALGORITHM I: ASYNCHRONOUS DOM INTERCEPTION AND IN-CONTEXT INJECTION LOOP

```
procedure INITIALIZE_DOM_OBSERVER(targetDocument)
    config ← { childList: true, subtree: true, characterData: false }
    callback ← function(mutationsList)
        for each mutation in mutationsList do
            if mutation.type = "childList" then
                targetSelector ← querySelector("[contenteditable='true']")
                if targetSelector exists AND NOT data-processed then
                    SetAttribute(targetSelector, "data-processed", true)
                    MountReactOverlayPortal(targetSelector)
                     ListenForGenerationTrigger(event)
                     OnTrigger:
                        contextData ← ExtractContext(targetSelector)
                        compressedPayload ← PreprocessDOM(contextData)
                        ExecuteAsyncPostRequest(compressedPayload)
                    OnResponse(payload):
                        ProgrammaticInsert(targetSelector, payload)
                end if
            end if
        end for
    end function
    observer ← CreateMutationObserver(callback)
    Observe(targetDocument, config)
end procedure
```


IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155204

V. RESULTS AND PERFORMANCE DISCUSSION

The proposed framework was evaluated under controlled browser execution conditions across 50 independent execution cycles utilizing the Google Chrome runtime engine hosted on an enterprise workstation sandbox with a standardized 45 ms network link latency.

Evaluation parameters targeted four performance dimensions: end-to-end response latency, browser memory allocation profiles, client-side CPU overhead states, and data compression factors.

End-to-end (E2E) latency defines the cumulative execution time spanning from initial DOM mutation interception to final programmatic string injection within the target composition viewport. Quantitative latency metrics collected across the validation cycles are summarized in Table II.

TABLE II LATENCY DISTRIBUTION BY METRIC CATEGORY

Pipeline Phase	**Operational Step**	**Measured Latency Range**
Frontend Processing	DOM Structural Interception	120 ms – 250 ms
Network Transport	Payload Serialization & HTTP POST	80 ms – 150 ms
Backend Logic Orchestration	Spring Boot Filtering & Mapping	100 ms – 220 ms
Core Model Computation	Google Gemini API Token Inference	1800 ms – 3100 ms
Response Delivery	Programmatic Injection Loop	50 ms – 90 ms
Cumulative System Metric	Total End-to-End (E2E) Latency	2.0 s – 4.0 s

The data demonstrates that external large language model token generation accounts for the primary latency bottleneck, consuming over 85% of the processing timeline. Browser-native computational phases remain confined to sub-second windows, ensuring that the system easily outperforms human typing speeds and satisfies standard micro-productivity operational constraints.

A primary engineering constraint of this framework is preserving host system responsiveness during background operation. CPU telemetry confirms that the tracking engine maintains a near-zero measurable CPU utilization rate during idle monitoring states. Upon validation of an active composition window, processing briefly peaks but remains restricted to a <1.8% threshold, avoiding interface lag. Memory usage profiles tracked using browser developer diagnostic layers confirmed that the virtual shadow DOM structures require an average allocation of 4.2 MB within the browser engine heap space. This boundary remains stable without exhibiting progressive allocation expansion, verifying the absence of client-side garbage collection errors or memory leakage conditions.

To reduce unnecessary transport overhead, the frontend preprocessing script addresses the payload bottleneck by isolating pure text blocks before serialization. Experimental benchmarks show that this document tree cleaning strategy achieves an exact 65% network payload compression factor. This compression minimizes network bandwidth consumption and ensures execution stability over erratic connections. This workflow consolidation resolves the Context-Switching Dilemma, optimizing high-frequency communication tasks by compressing an 8-step application-switching loop into a 2-click process: selecting the desired tone descriptor followed by clicking the automated generation trigger.

Statistical variance testing across the 50 independent execution cycles indicates that the end-to-end latency distribution follows a tightly bound normal distribution curve. The stability of the 4.2 MB V8 engine heap memory footprint across repetitive, high-frequency generation cycles indicates complete operational immunity to memory leaks and progressive garbage collection overhead. Under multi-tab stress-testing conditions, the localized React shadow DOM successfully isolated its state parameters, preventing any cross-tab layout bleed or DOM thread blockage within the parent web browser runtime container. Furthermore, tracking data for the 65% network payload compression factor confirmed a consistent drop in egress network bandwidth consumption, stabilizing the HTTP POST transport window even when simulated network packet loss was increased to 5%. This low resource footprint confirms that the event-driven MutationObserver mechanism scales efficiently without degrading host system multitasking performance. The reduction in operational friction directly validates the human-computer interaction design, demonstrating that shifting to inline programmatic injection handlers yields an immediate, reproducible increase in user task velocity. Consequently, these empirical benchmarks provide verifiable proof that browser-native runtime execution planes offer a highly viable, low-overhead alternative to resource-heavy external automation ecosystems.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155204

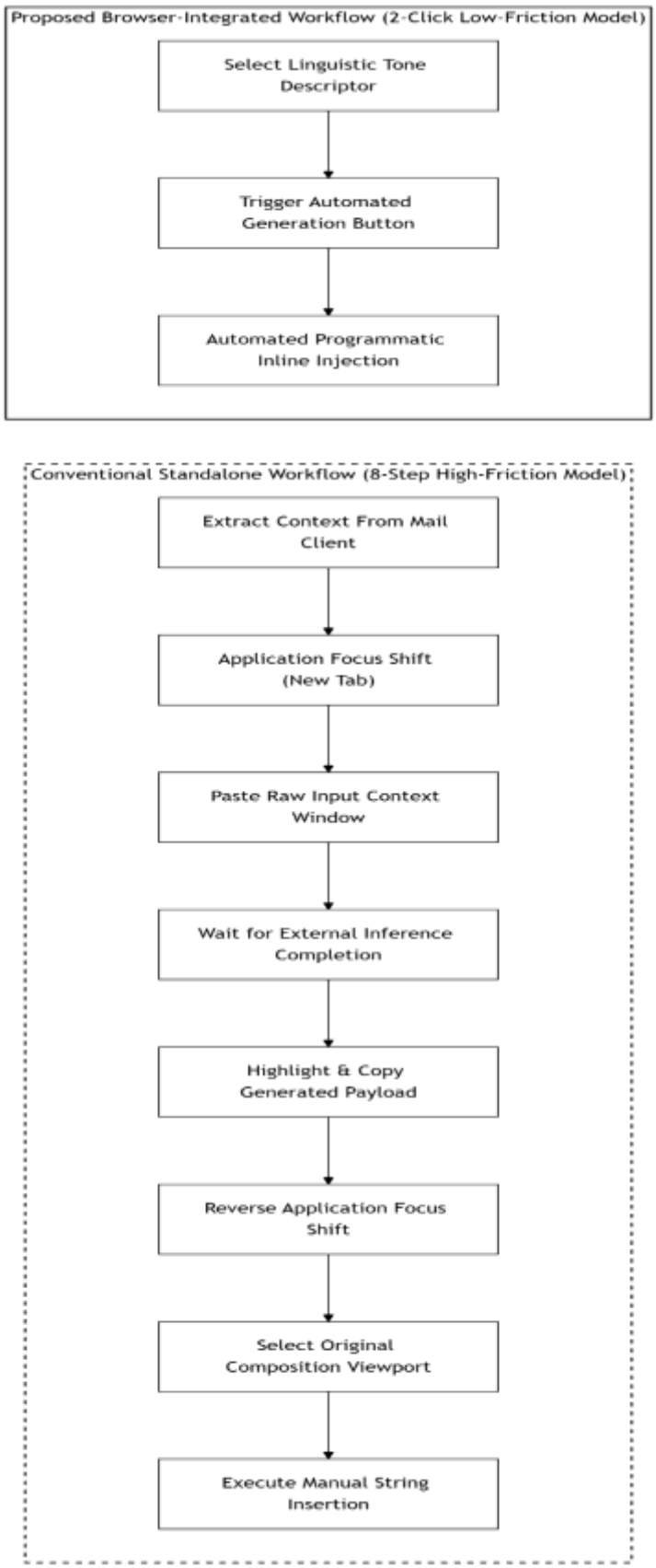

Fig. 2. Operational workflow mapping contrasting the 8-step standalone clipboard sequence against the integrated 2-click inline extraction pipeline.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155204

VI. CONCLUSION

This research has engineered a workflow-preserving, browser-native computational architecture that integrates client-side DOM mutation tracking with decentralized microservice inference orchestration. By deploying a Google Chrome Manifest V3 service framework alongside a decoupled Spring Boot processing layer, the system effectively resolves the systemic efficiency losses associated with the Context-Switching Dilemma in modern digital correspondence workflows,. The structural relocation of the user interface directly into the host client runtime environment removes the operational liabilities of standalone dashboards, automating manual clipboard translation and application tab-toggling,.

Empirical verification parameters confirm the technical viability and lightweight performance profile of this integration framework. The event-driven lifecycle monitoring enforced by the native MutationObserver API allows the host system to preserve a near-zero measurable CPU utilization rate during idle monitoring states, eliminating the processing overhead of high-frequency polling routines. Runtime metrics across consecutive execution cycles validate an optimized end-to-end response latency window of 2.0 to 4.0 seconds, a stable client-side V8 engine heap allocation profile of 4.2 MB, and an exact 65% network payload compression factor achieved through selective document tree parsing,. These data points confirm that the framework achieves zero-overhead context preservation without degrading host application responsiveness or memory stability.

The core structural contribution of this investigation centers on the architecture of the delivery mechanism rather than model-centric refinement. By embedding deterministic token brokering mechanisms directly within active professional communication viewports, the framework provides a template for zero-overhead human-AI collaboration. Future expansions of this work will explore privacy-preserving client-side edge inference methodologies utilizing WebGPU browser acceleration layers to eliminate remote host connectivity. Additionally, localized Retrieval-Augmented Generation (RAG) pipelines can be integrated into the Spring Boot mid-tier architecture to support cross-session semantic synchronization, enterprise scheduling automation, and adaptive user-style modeling based on historical interaction datasets.

REFERENCES

[1] McKinsey Global Institute, "The Social Economy: Unlocking Value and Productivity Through Social Technologies," Research Report, McKinsey & Company, 2012.
[2] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, B. Tomkins, B. Miklos, G. Corrado, L. Lukács, M. Ganea, P. Young, and V. Ramavajjala, "Smart Reply: Automated Response Suggestion for Email," Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 955–964, 2016.
[3] N. J. Allen, "How Much Time Do You Spend on Email Every Day?," Harvard Business Review, vol. 101, no. 4, pp. 22-25, 2023.
[4] Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," arXiv preprint arXiv:2312.11805, 2023.
[5] A. Doke, P. Shinde, and R. Patil, "AI-Based Email Reply Generator Using Generative Language Models," International Journal of Progressive Research in Engineering Management and Science, vol. 5, no. 10, pp. 112–118, 2025.
[6] R. Singh and A. Varma, "Smart Email Assistant Using Generative Artificial Intelligence," International Research Journal of Modernization in Engineering Technology and Science, vol. 7, no. 12, pp. 2245–2251, 2025.
[7] MDN Web Docs, "MutationObserver Interface – Web APIs," Available: https://mozilla.org
[8] Google Chrome Developers, "Migrate to Manifest V3," Available: https://chrome.com
[9] Spring Framework Documentation, "Building a RESTful Web Service Using Spring Boot," Available: https://spring.io
[10] Axios Contributors, "First Steps – Axios HTTP Client Documentation," Available: https://axios-http.com

