Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Beyond Keywords: Intent-Aware Message Routing for Gmail and Slack Using Large Language Models

Ijegwa Acheme

david.acheme@wigweuniversity.edu.ng

Wigwe University

Emmanuella Amukwuo

Miva Open University

Ayodeji Makinde

Edo State University Uzairue

Shangbum Faveren

Universidade Federal do Pará

Chukwuka Obionwu

Wigwe University

Research Article

Keywords: conversational AI, email automation, intent detection, large language models, Slack integration

Posted Date: October 30th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11333689/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Beyond Keywords: Intent-Aware Message Routing for Gmail and Slack Using Large Language Models

Abstract

In many corporate business organizations today, message misrouting has become a persistent problem among professional team members, because communications frequently reach unintended recipients and will therefore require manual redirection, this leads to delays in response times and disruption to workflow. There are existing message automation tools designed to handle this like Gmail filters, Slack workflows, Zapier, and Make but they rely largely on keyword or condition matching, which can perform poorly given the natural variability of message phrasing in real communication. This study presents the design and implementation of an intelligent message routing and triage platform that applies natural language classification to automatically process and route incoming Gmail and Slack messages. The platform integrates the LLaMA 3.1 large language model, which was accessed via the Groq API, with the Gmail and Slack APIs, enabling semantic rather than keyword-based classification. The design of the system allows users to define routing rules in plain English through a web-based dashboard, while a background polling service monitors connected inboxes every five minutes for new messages. Each incoming message is classified according to semantic intent against user-defined rules, triggering one of three actions (forwarding, automated reply, or ignoring) or, where no rule matches or the classifier flags the message for user action, escalation to a human reviewer, with every event logged for auditability. The system was developed iteratively across four development cycles and implemented as a full-stack TypeScript application using Next.js, Express.js, and MongoDB. The evaluation comprised functional testing, integration testing, and a preliminary feasibility check of the classifier. Functional testing achieved a 100% pass rate across all component categories (34 tests). The feasibility check used 14 author-constructed messages covering four of the six supported rule categories: 12 were matched to the intended rule (85.7%; 95% Wilson confidence interval 60.1–96.0%) and two were not matched and were escalated to human review, whereas a simulated keyword-matching baseline matched 5 of 14 (35.7%; 16.3–61.2%). Because the set is small, was written to avoid the rule wording, and was labelled by the authors, these figures illustrate the system's behaviour and do not establish comparative accuracy. Both unmatched messages involved indirect scheduling language not explicitly captured by rule conditions, indicating that classification performance remains sensitive to rule specificity. The key contribution is a documented system design, including an abstain-and-escalate decision rule, for embedding large language model classification into multi-channel business productivity tools, with particular relevance for Nigerian and African small and medium-sized enterprises seeking affordable, accessible, and intelligent communication automation.

Keywords: conversational AI; email automation; intent detection; large language models; Slack integration

1 Introduction

Communication in the workplace has become more immediate, and more fragmented over the last decade. Gmail and Slack now handle much of the routine communication that takes place in organisations, of any size, with Gmail serving over 1.8 billion active users worldwide (Statista 2023) and Slack reporting over 20 million daily active users (Slack Technologies 2023). These numbers tell us how much we are communicating digitally, but also highlight a less evident cost of operation: the number of messages that workers are forced to constantly sort, interpret and redirect (Zeldes et al. 2007). Someone must decide the purpose, urgency and who should act upon incoming messages from customers, suppliers, colleagues and business partners – an activity often called message triage, and one found to take up a measurable portion of the working day through email-driven interruption and re-immersion costs (Jackson et al. 2003). A 2012 report from the McKinsey Global Institute estimated that knowledge workers spent some 28% of their time managing email (Chui et al. 2012), and a 2019 study by Adobe found a broadly similar pattern (Abramovich 2019), suggesting that the growth in workplace communication tools has not in itself solved the administrative burden of inbox management.

In Nigeria and other African markets where a lot of tech firms and small businesses run with very limited team members, the problem is worse. For example, Lagos has become one of the leading technology hubs in Africa (GSMA 2022) and Nigerian organisations are increasingly using cloud-based collaboration platforms for day-to-

day coordination (Zoho Corporation 2024). These organisations don't always have dedicated admin or customer support staff, so founders, developers, or ops people have message routing/re-routing added to their responsibilities. This clearly has negative effects on overall productivity because it can delay responses and pull staff focus from core duties as message volumes increase – a cost that is amplified by the fact that a professional receives an average of 121 business emails daily (Radicati Group 2023). Also, for small businesses that serve customers online, slow or misdirected replies are also a customer-experience problem and not only an internal productivity problem (Acheme et al. 2023).

The existing platform's features only partially address this. Gmail filters can act on the presence of certain words or metadata in a message, but routine day to day communication are not always systematic enough that exact keyword rules can be reliably maintained (Galitsky 2020; Shi and Mariano 2025). Classes of intent are typically only weakly correlated with any fixed set of keywords. A user with a login problem might write "I can't access my account", "my login is broken" or just "help" - a filter written around one expression might miss the others, even if the underlying intention is identical. A related limitation is also found in Slack automation because slack is mostly event- and condition-driven and does not interpret the meaning of messages. More sophisticated automation tools such as Zapier and Make offer greater flexibility, but their setup demands technical expertise, and financial burdens because of subscription fees. For small businesses in Africa, paying subscriptions domiciled in US dollars may poses a significant challenge.

Recent advances in large language models (LLMs) offer a different technical possibility. Meta's LLaMA family of models, for example, can evaluate a message by intent, rather than looking for specific keywords, allowing for a routing system that can recognise different ways of saying the same thing and then act accordingly. What remains absent from current practice is an accessible platform that interprets message intent, applies routing rules without code, and serves organisations that cannot justify the cost or complexity of enterprise-grade automation software. This paper addresses that gap by presenting the design, implementation and preliminary evaluation of an intelligent message routing and triage platform that integrates LLM-based semantic classification with the Gmail and Slack application programming interfaces (APIs), proposing an affordable design alternative to keyword-dependent automation for small and growing organisations, along with a preliminary evaluation and feasibility check of its behaviour.

2 Related Work

2.1 Rule- and Condition-Based Commercial Automation Platforms

Before defining the proposed design, it was necessary to examine tools and approaches already used for message automation. The systems reviewed addresses a range of the routing problem, from simple inbox filters to enterprise workflow platforms. Their capabilities are useful, but the comparison reveals the same limitations in semantic interpretation, cross-channel consistency, cost, and configuration complexity.

2.1.1 Gmail Native Filters

Gmail offers an integrated filter feature that enables users to set rules based on message attributes such as the sender address, keywords in the subject-line, the text of the body, whether there is an attachment and its size. then matching messages can be labelled, archived, forwarded, starred or deleted (Google n.d.); It's included with the service at no extra cost, making it one of the most accessible forms of email automation. And that same simplicity is its problem: filters look for metadata and exact text patterns, not the intent behind a message. For instance, take two messages about a billing problem. A rule based on the word 'payment' will not reliably catch 'I have not been charged correctly this month'. This sensitivity to phrasing undermines real inbox reliability. Also the rules only work inside gmail, they don't have reach into slack or other channels.

2.1.2 Slack Workflow Builder

Slack Workflow Builder is a paid no-code environment (Pro, Business+, and Enterprise plans) for designing automated sequences inside a workspace (Slack Technologies n.d.). A workflow begins with a specific trigger such as a scheduled time, form submission, channel join, or an event from a connected third-party service like Salesforce, Asana, GitHub, or PagerDuty, and then runs through a series of steps such as posting a message or

collecting information via a form. Workflow Builder supports conditional branching (up to 15 conditions per workflow), but does not interpret the semantic content of a message. It can react to a post appearing in a given channel, but it cannot determine whether that post concerns billing, technical support, or another function. Automation still lives in Slack, so email triage is handled by an entirely separate tool.

2.1.3 Zapier

Zapier is a commercial integration service connecting several thousand applications through automated workflows ("Zaps", with both Gmail and Slack supported as first-party integrations (Zapier Inc. 2026). This cross-platform capability gives it more flexibility than native tools, but its routing logic remains condition-driven: a Zap fires on configured text or metadata matches, rather than evaluating the broader meaning of a request. The paid tier of Zapier, the most common low-level paid option, costs around $29.99 per month (or about $20 per month if paid annually), with a limited number of monthly tasks, billed in US dollars, a model that has increased in price since 2024 and may be difficult for Nigerian and other African small and medium-sized enterprises (SMEs) to sustain as a recurring foreign-currency expense.

2.1.4 Make (formerly Integromat)

Make offers a visual, node-based environment for building multi-step workflows across many applications including Gmail and Slack, with more elaborate conditional branching available through its scenario designer (Make Technologies 2026). Make's entry paid tier is comparatively cheaper at around $9-11 per month for a similar operation allowance. But the core reliance on conditions and pattern matching is still the same as Zapier and building non-trivial scenarios still takes a fair bit of technical familiarity, even with the visual interface.

2.1.5 Microsoft Power Automate

Microsoft 365 ecosystem's enterprise automation platform, Power Automate, has a broad selection of connectors, conditional logic, and approval workflows (Microsoft n.d.). Among the rule-based platforms reviewed, it provides the widest array of enterprise workflow functions, but it mostly executes the logic that users explicitly configure. Semantic intent classification is not a primary focus of its routing model. It assumes an organization that can manage the Microsoft 365 environment. It's not really suitable for organisations built around Google Workspace, because it's Outlook-centric rather than Gmail as a first party channel.

2.2 Research on Automated Ticket Classification and Routing

Beyond commercial platforms, a body of research addresses automated classification and routing of support tickets and similar free-text messages, offering a more direct scholarly comparison point for the present work.

Early work by Montgomery and Damian (2017) used feature engineering over structured ticket metadata to predict support-ticket escalations in large software organisations, demonstrating that escalation behaviour is learnable but relying on structured fields rather than free-text semantic understanding. Marcuzzo et al. (2022) proposed a multi-level hierarchical approach to ticket classification that explicitly addresses class imbalance across support categories, improving on flat classification schemes but still depending on supervised training over a fixed, labelled taxonomy. Liu et al. (2023) introduced Ticket-BERT, a fine-tuned transformer model for labelling incident-management tickets, moving beyond keyword statistics toward contextual language representations; however, the approach requires domain-specific fine-tuning on a labelled corpus and offers no mechanism for a non-technical user to define or adjust routing logic directly.

Most closely related to the present work is TickIt (Liu et al. 2025), an LLM-powered ticket-escalation framework deployed on ByteDance's Volcano Engine cloud platform, which performs topic-aware, dynamic, and relationship-driven escalation using category-guided fine-tuning of LLMs. TickIt demonstrates that LLM-based classification can outperform static rule- or keyword-based ticket handling at production scale. It differs from the present work in two important respects: it operates within a single enterprise's internal support-ticket corpus rather than across independent, off-the-shelf inboxes and workspaces such as Gmail and Slack, and it depends on continuous supervised fine-tuning over a large proprietary dataset rather than user-authored, plain-English rules — a dependency that places it out of reach for small organisations without dedicated machine-learning capacity.

What Ticket-BERT and TickIt share, despite their gains over rule-based systems, is a continued reliance on labelled data and fine-tuning infrastructure. This dependency motivates a closer look at an alternative line of research: zero-shot and few-shot prompting approaches that classify messages against categories they were never explicitly trained on, the family of techniques this project adopts in an example-augmented prompting form (Section 4.2.4).

2.3 Theoretical Foundations of Zero-Shot and Example-Augmented Intent Classification

Academic research on intent classification has developed significantly over the past decade, moving progressively toward approaches that require less category-specific labelled data Zero-shot intent classification early work was focused on finding ways to identify categories without having dedicated training examples for each category: Liu et al. (2019) proposed a capsule-network architecture that reconstructs unseen intent representations from known ones, and demonstrated that intent categories could be classified without per-category labelled examples; however, this approach still required a structured training phase over a fixed set of seen intents and did not generalise to categories dynamically introduced by end users.

Zero-shot learning as a broader paradigm was also formalised outside the language domain. Xian et al. (2019) established influential benchmarks for zero-shot learning in computer vision, showing that models trained on seen categories could generalise to unseen ones through semantic attribute descriptions. Although this work addresses image rather than text classification, the underlying principle — that a category can be recognised from its description rather than from labelled examples of it — is the same principle later realised for text classification through LLMs, and it is in this sense that the present project draws on it.

Brown et al. (2020) demonstrated with GPT-3 that sufficiently LLMs could perform classification and other tasks directly from natural language prompts, including in zero-shot and few-shot settings, without any gradient updates to the underlying model. Wei et al. (2022) subsequently showed that instruction-tuned models exhibit substantially improved zero-shot performance on unseen task types, including classification-style tasks, when fine-tuned on a broad mixture of instruction-described datasets. Together, these findings provide the theoretical basis for the approach adopted in this project: using a pre-trained, instruction-tuned LLM with a structured prompt to classify messages against user-defined plain-English rule conditions, without any task-specific fine-tuning.

More recent work has examined which properties of a prompt actually drive this zero-shot and few-shot behaviour. Min et al. (2022) found that in in-context learning, the ground-truth pairing between an example and its label matters far less than expected — replacing correct labels with random ones barely affects performance — while the label space and format specified by the prompt matter considerably. This finding is relevant to the present design, which combines clearly specified rule labels and natural-language descriptions with a small number of example phrasings per category; the separate contribution of those examples was not isolated in this study (Section 5.3.4).

Compared to supervised intent classification systems — including the ticket-routing approaches reviewed above (Liu et al. 2023; Liu et al. 2025; Marcuzzo et al. 2022; Montgomery and Damian 2017) — the prompt-based approach adopted here trades peak accuracy for flexibility and zero data-collection overhead. A supervised classifier trained specifically on routing categories might achieve higher accuracy within those categories, but would require substantial labelled examples per category and would not adapt dynamically when a user adds or modifies a rule. For the target use case — small teams defining their own routing policies without technical support — prompt-based classification is a plausible and practical trade-off, although this study does not test it against supervised baselines.

2.4 The Research Gap

Looking at both the commercially available message routing software applications and existing research works, reveals that the missing link is not an absence of automation, or even an absence of semantic classification research, but a divide between the two. While commercially available tools are accessible and, in some cases (native Gmail and Slack features) free, They remain platform-specific and condition-based. The higher tools such as Zapier, Make, and Power Automate yield better results by extending reach across channels but they still depend on predefined logic cost associated with configuration overhead. Academic systems like Ticket-BERT and TickIt

demonstrates that semantic, LLM-based classification outperforms keyword matching significantly, but these systems require labelled training data, fine-tuning infrastructure and machine learning (ML) engineering capacity that are feasible for a large technology company but not for a resource-constrained SME. No reviewed system, commercial or academic, combines LLM-based semantic intent classification, a single set of user-authored plain-English rules that require no fine-tuning, multi-channel coverage of Gmail and Slack, and a cost and configuration profile suitable for the Nigerian and African SME context. Figure 1 summarises these differences and provides the practical basis for the proposed platform.

	Gmail Filters	Slack Workflows	Zapier	Make	Power Automate	Academic LLM/ML Systems	Proposed System
Semantic (LLM-based) intent classification	✗	✗	✗	✗	✗	✓	✓
Multi-channel support (Gmail + Slack)	✗	✗	✓	✓	~	✗	✓
Plain-English rule definition (no code)	✗	✗	✗	✗	✗	✗	✓
No ML fine-tuning / labelled data required	–	–	–	–	–	✗	✓
Affordable for resource-constrained SMEs	✓	✓	✗	~	✗	✗	✓
Human-in-the-loop escalation option	✗	✗	✗	✗	✓	~	✓

✓ = Yes ✗ = No ~ = Partial / Limited – = Not applicable

Fig. 1 Comparison of existing solutions with the proposed system

3 System Design and Implementation

3.1 System Architecture

The platform is built on a client–server architecture with a Next.js frontend, an Express.js backend (written in TypeScript), a MongoDB data store, and integrations with the Gmail API, the Slack Web API, and the Groq API. The web interface does not talk directly with these external services. Instead, it makes authenticated representational state transfer (REST) requests to the backend, which enforces business rules, reads and writes application data, and orchestrates all calls to external APIs.
The background polling service decouples message retrieval from the normal request-response cycle. This process is independent of any user activity on the server and periodically checks each connected account of the user. If it finds a new message, it starts the classification and routing sequence without the user needing to be actively using the dashboard. The resulting data flow is summarised in Fig. 2.

At a high level, information moves through the architecture (Fig. 2) as follows:

1. The registered user manages connections and routing rules through the Next.js dashboard.
2. Authenticated Hypertext Transfer Protocol (HTTP) requests carry dashboard operations to the Express.js backend API.
3. The backend uses the Mongoose object–document mapper (ODM) to store and retrieve users, rules, tokens, and activity records in MongoDB.
4. The polling service retrieves the active rule set for each user from the backend.

5. At fixed intervals, the polling service queries the Gmail and Slack APIs for new messages belonging to connected accounts, and later executes the resulting routing action through the same APIs.
6. Message content and the active rule set are submitted to the LLM service (LLaMA 3.1 via the Groq API) for classification, which returns an intent, an urgency level, and a confidence score.
7. The platform writes the classification outcome and the action taken to the activity log.

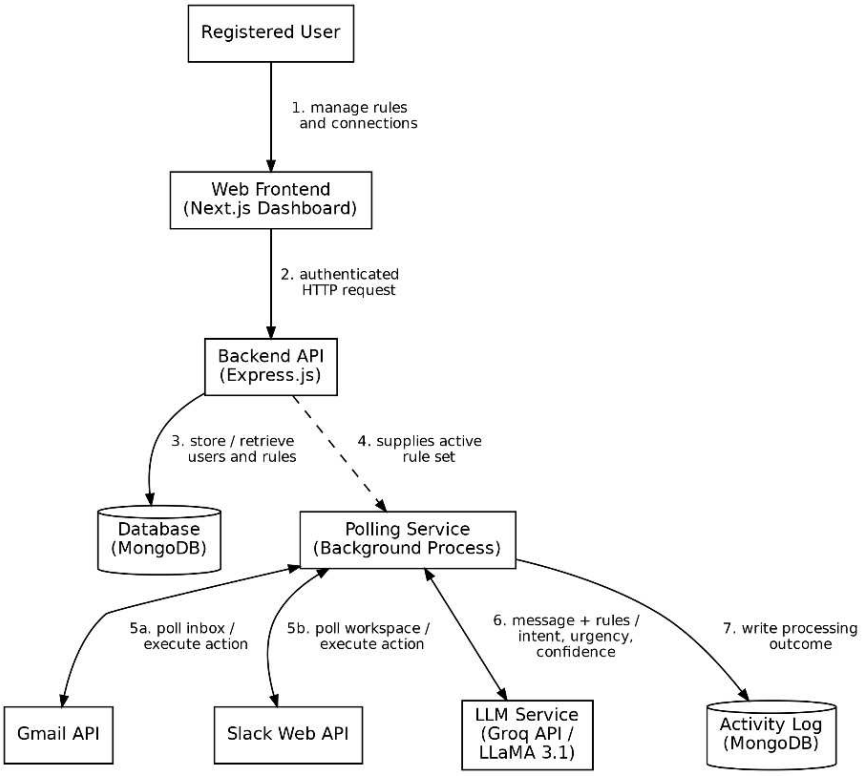


Fig. 2 System architecture and data flow of the intelligent message routing platform

3.2 System Components

Table 1 summarises the responsibilities of each architectural component.

Table 1 System components and responsibilities

Component	Technology	Responsibility
Web frontend	Next.js (React, TypeScript)	User interface for account management, channel connection, rule management, and activity log review
Backend API	Express.js (TypeScript)	Authentication, rule storage, polling orchestration, action execution
Database	MongoDB with Mongoose	Storage of user accounts, routing rules, and activity logs
Authentication	JavaScript Object Notation (JSON) Web Token (JWT) + bcryptjs	Secure user login and session management
Gmail integration	Google Gmail API	Reading inbox messages, forwarding, and sending replies
Slack integration	Slack Web API	Reading workspace messages and posting routing responses
Classification engine	Groq API / LLaMA 3.1	Intent classification and routing-decision generation
Polling service	Node.js background process	Periodic retrieval of Gmail and Slack messages
Activity logger	MongoDB (Activity model)	Recording all system actions for user transparency
Billing module	Custom implementation	Subscription management and access control

Figure 3 complements the data-flow view in Fig. 2 by grouping the components listed in Table 1 into layers according to their responsibility. The presentation layer (the Next.js dashboard) handles account and channel connection, rule management, review of the activity log and the human-review queue. The application layer

(Express.js) contains authentication, rule management, billing, the polling service, the classification engine and the action executor. Persistent data is held in MongoDB, which is kept separate from the external services (the Gmail API, the Slack Web API and the Groq API). The figure shows that the dashboard communicates only with the backend, and that the backend alone mediates access to the database and to the external services. This separation supports access control and makes it easier to add further channels (section 7.2).

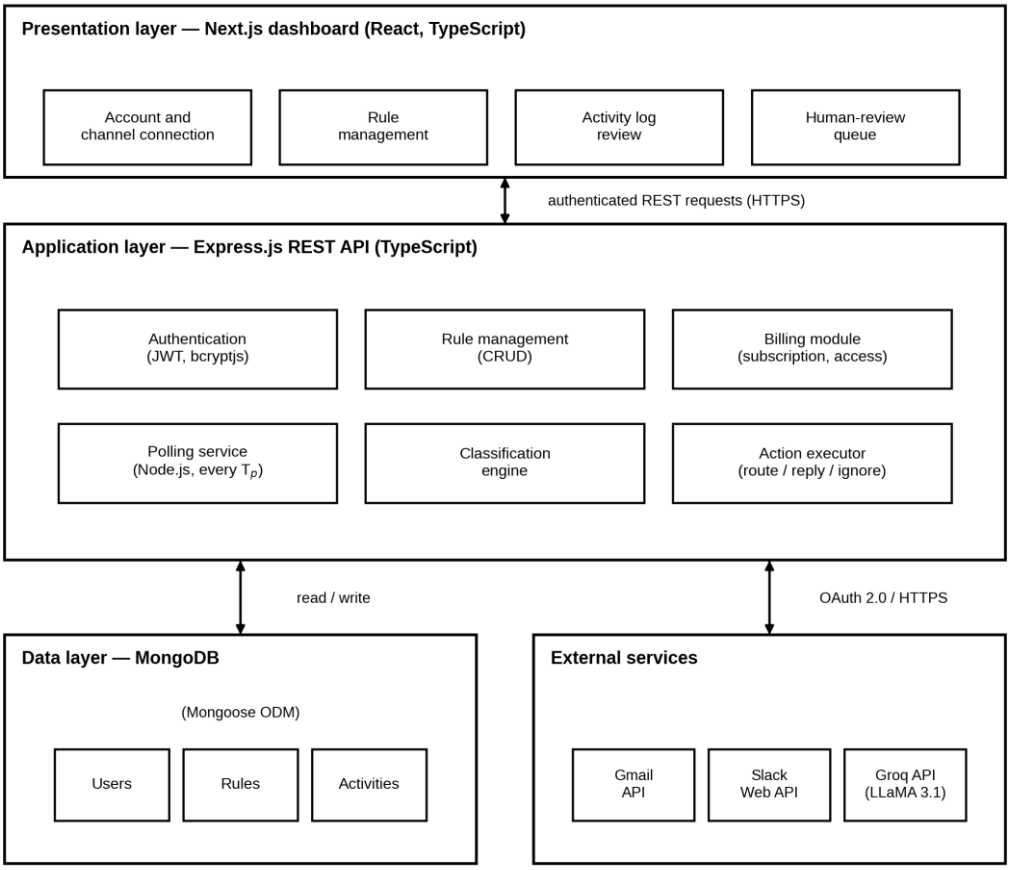


Fig. 3 Layered architecture of the intelligent message routing platform, showing the presentation, application, data and external-service layers

3.3 Message Classification and Routing Model

Using Eq. (1), we express the classification and routing behaviour of the polling service formally. Let M denote an incoming message and let $R_u = \{r_1, r_2, \ldots, r_n\}$ denote the set of active plain-English rule conditions defined by user u, where each rule r_i is associated with an action $a_i \in$ {route, reply, ignore} and, where applicable, a destination or reply text.

The classification engine implements a function

$$f: M \times R_u \rightarrow (i^*, g, c) \tag{1}$$

where $i^* \in \{1, \ldots, n\} \cup \{\emptyset\}$ is the index of the matched rule ($\emptyset$ denoting no match against any active rule), $g \in$ {low, medium, high} is the assigned urgency level, and $c \in [0, 1]$ is a confidence score. f is realised through a single LLM inference call in which M and R_u are encoded into a structured prompt; no gradient update or fine-tuning of the underlying model is performed for any individual classification.

Let $h \in$ {true, false} denote the requiresUserAction flag returned with the classifier output. The routing decision $d(M)$ applied by the polling service is:

$$d(M) = \begin{cases} \text{execute } a_{i^*}, & \text{if } \; i^* \neq \emptyset, h = \text{ false and } g \neq \text{ high} \\ \text{escalate to human-review queue, otherwise} \end{cases}$$

That is, a message is acted on automatically only when it is matched to a rule, the model does not flag it for user action, and its urgency is not high; all other cases — including messages that match no active rule — are deferred to a human reviewer rather than acted on speculatively. In the implemented system, the confidence score c is recorded in the activity log but is not compared with a numerical threshold; a configurable threshold θ is left as an extension (section 7.2).

The polling service also bounds the delay between message arrival and the start of processing. If T_p denotes the polling interval (five minutes in the current implementation) and t_a denotes a message's arrival time, the worst-case latency $L(M)$ before that message is first considered for classification satisfies:

$$0 \leq L(M) \leq T_p \tag{2}$$

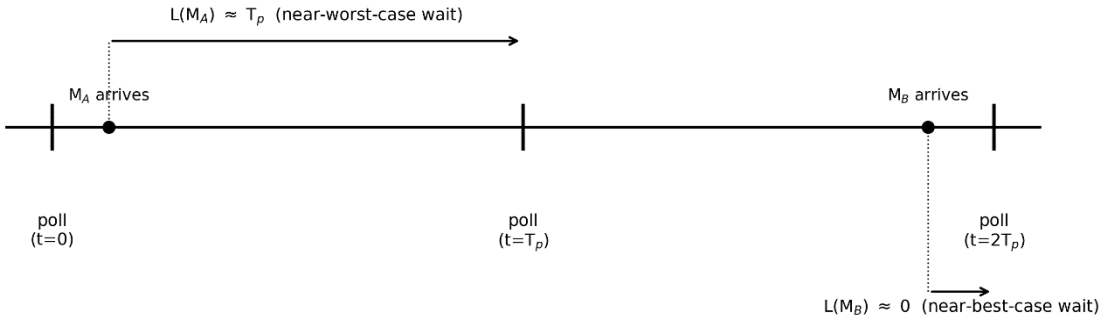


Fig. 4 Waiting time between message arrival and the next poll under fixed-interval polling: a message arriving just after a poll waits almost one full polling interval, whereas one arriving just before the next poll waits almost no time

From Eq. (2) and Fig. 4, a message arriving immediately after a polling cycle must wait at most one full interval before the next cycle detects it, while one arriving immediately before a cycle is detected almost immediately.

3.4 Use Case Description

The use-case model distinguishes between actions initiated directly by a registered user and processes carried out automatically by the platform once configured, as shown in Fig. 5.

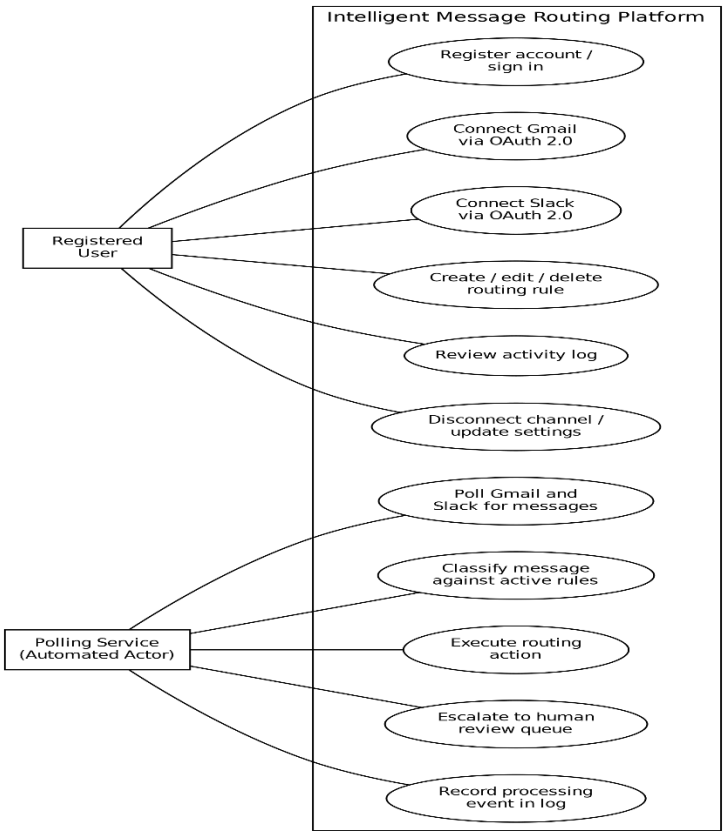


Fig. 5 Use case diagram of the intelligent message routing platform

From Fig. 5, the primary actor is a registered user, such as a business owner, team manager, or professional.

User-initiated use cases:

- Registering an account and signing in to the platform
- Connecting a Gmail account via Open Authorization (OAuth) 2.0
- Connecting a Slack workspace via OAuth 2.0
- Creating, editing, activating, deactivating, and deleting routing rules expressed in plain English
- Reviewing the activity log for processed messages, classifications, and completed actions
- Disconnecting communication channels and updating account settings

System-initiated use cases (executed automatically after configuration):

- Polling the Gmail inbox and connected Slack workspace at the fixed interval T_p
- Classifying each new message against the user's active rules via $f(M, R_u)$
- Applying the routing decision $d(M)$: forwarding, replying, or ignoring
- Escalating unmatched or low-confidence messages to the human-review queue
- Recording the message-processing event and resulting action in the activity log

3.5 Database Design

The data layer is implemented in MongoDB, a document-oriented non-relational (NoSQL) database, with Mongoose providing schema definitions, validation, and query access. Three collections are central to the application — Users, Rules, and Activities — keeping account and integration data separate from user-defined policy and from the audit history of processing events.

3.5.1 User Model

Table 2 summarises the fields of the User collection, which holds the account, authentication, integration, subscription, and polling information required for each registered user.

Table 2 User model fields

Field	Type	Description
_id	ObjectId	Unique identifier (auto-generated)
email	String	User's email address (unique, required)
password	String	Bcrypt-hashed password
gmailTokens	Object	OAuth 2.0 access and refresh tokens for Gmail
slackTokens	Object	OAuth 2.0 access and refresh tokens for Slack
gmailConnected	Boolean	Whether Gmail integration is active
slackConnected	Boolean	Whether Slack integration is active
subscriptionStatus	String	Current subscription status
lastGmailPoll	Date	Timestamp of the last Gmail polling cycle
lastSlackPoll	Date	Timestamp of the last Slack polling cycle
createdAt	Date	Account creation timestamp

3.5.2 Rule Model

Table 3 presents the Rule collection, which stores the plain-English routing instructions r_i owned by each user, together with the associated action a_i and any destination or reply text required to execute it.

Table 3 Rule model fields

Field	Type	Description
_id	ObjectId	Unique identifier
userId	ObjectId	Reference to the owning User document
condition	String	Plain-English rule condition, e.g. "if someone asks about billing"
action	String	Action type: route, reply, or ignore
routeTo	String	Email address or Slack channel to route to (for route action)
replyMessage	String	Automated reply text (for reply action)
isActive	Boolean	Whether the rule is currently active
createdAt	Date	Rule creation timestamp

3.5.3 Activity Model

Table 4 presents the Activity collection, which stores a record of every message-processing event, including the channel, classification outcome (i^*, g, c), matched rule, completed action, and whether human attention was required.

Table 4 Activity model fields

Field	Type	Description
_id	ObjectId	Unique identifier
userId	ObjectId	Reference to the owning User document
channel	String	Source channel: gmail or slack
sender	String	Sender email address or Slack user identifier (ID)
subject	String	Message subject line (Gmail) or channel name (Slack)
intent	String	Classified intent returned by the LLM
urgency	String	Urgency level g: low, medium, or high
matchedRuleId	ObjectId	Reference to the matched Rule (i^*), where applicable
actionTaken	String	The action executed by the system
routedTo	String	Recipient of a forwarded message, where applicable
requiresUserAction	Boolean	Whether the message was flagged for human review
confidence	Number	Classification confidence score $c \in [0, 1]$
createdAt	Date	Timestamp of the processing event

3.5.4 Entity–Relationship Description

The three collections are linked via document identifiers as shown in Fig. 6. One User document can have many Rule documents and many Activity documents. The matching classification's Activity record has a reference to the matching Rule, creating a many-to-one relationship from activities to rules. This architecture allows the polling

service to query for all active rules for a user at once, and then relate each processing event back to the owning account and the rule that led to the routing decision.

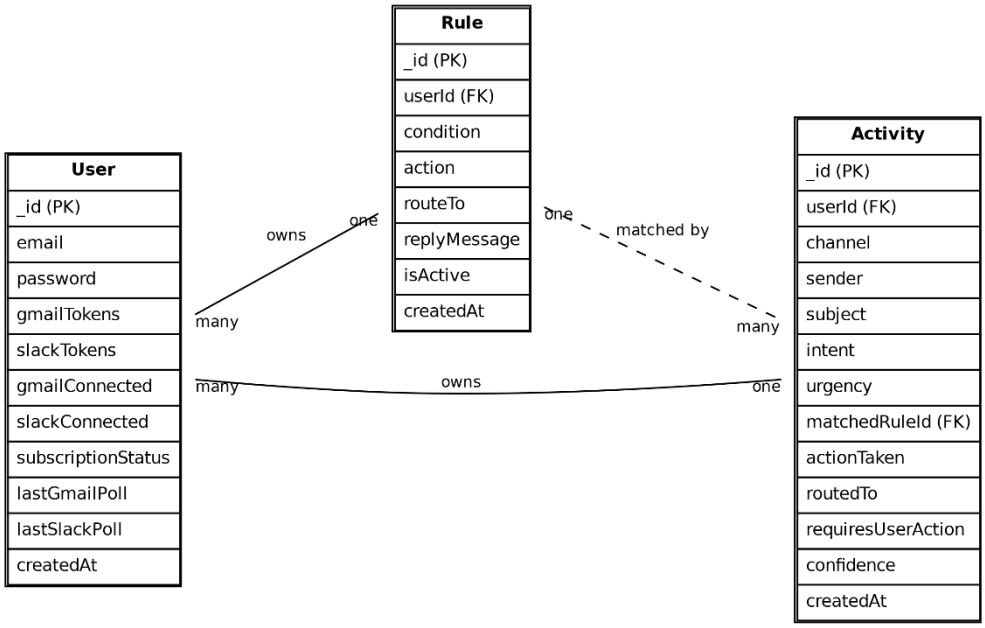


Fig. 6 Entity–relationship diagram of the User, Rule, and Activity collections

4 Implementation

4.1 Development Environment and Technology Stack

The platform has been built as a full stack TypeScript application following the architecture described in section 3.1. The dashboard is a browser-based application built with Next.js. The backend exposes a REST API using Express.js. Data persistence is provided by MongoDB using Mongoose. Using TypeScript across the frontend and backend helped to minimise data-structure mismatches at the integration boundaries, especially between the dashboard, Express routes, MongoDB models and third party API responses, and made it easier to maintain shared interfaces as the schema evolved. The external integration used three official or first-party client libraries: the Node.js googleapis package for Gmail (OAuth 2.0 flow, token refresh, and typed endpoint access), the @slack/web-api package for Slack, and the Groq software development kit (SDK) for LLM inference against LLaMA 3.1. The development tooling (editor, hot-reload via Nodemon/ts-node, Git-based version control) was standard practice for a TypeScript monorepo and is omitted here as it has no bearing on the platform's design or results.

4.2 Implementation of Core Modules

4.2.1 Authentication

Authentication was done first because all connections, rules, and activity records are scoped to an identified user. When you sign up, bcryptjs hashes the password you provide before persisting it to MongoDB; never saving plaintext passwords. When logging in, the backend compares the provided password to the stored hash and, if it matches, returns a signed JWT with the user's database id. The token is then required in the Authorisation header of protected routes and is verified by middleware to check the signature before the request is allowed to proceed.

4.2.2 Gmail and Slack Integration

Both channels are based on the OAuth 2.0 authorization-code flow, with differences only in the endpoints scope and conventions. In the case of Gmail, the connection option sends the user to Google's consent screen, and when approved, the backend exchanges the authorisation code returned for access and refresh tokens, which are linked to the user's record and the polling service then calls the messages.list endpoint at fixed interval T_p to identify

unread messages. The Slack integration works the same way through the consent flow from Slack itself and the conversations.history endpoint, with per-channel polling state to avoid re-processing already-seen messages.

4.2.3 Rule Management

The rules module exposes REST endpoints for creating, reading, updating and deleting the routing instructions r_i defined in section 3.3. All operations check that the authenticated user owns the rule being modified before accepting a modification. The user states a condition in plain English via the dashboard and provides an action $a_i \in$ {route, reply, ignore} and, if necessary, a destination address, channel or reply text.

4.2.4 Message Classification and Routing Engine

The module implements the function f(M,R_u) and the decision rule d(M) formalised in section 3.3 and is the main technical contribution of the platform. The polling service calls the classifier with every newly retrieved message and the active rule set of the caller. We use the Groq API to send a structured prompt to meta-llama/llama-3.1-8b-instant that includes sender, subject, a truncated message body (up to 400 chars), the active rules, and explicit instructions for the desired output-format, as well as a small number of semantically varied example phrasings per category to encourage intent-based rather than literal keyword matching. The classifier is therefore best described as example-augmented prompting rather than strictly zero-shot classification. The inference temperature was set to 0.1 to minimise variance in the outputs for repeated calls on similar inputs. The classifier returns a confidence value as part of the structured model output; however, no separate numerical confidence threshold was implemented in the routing logic. Routing and escalation decisions were instead based on the model's matched-rule output, the requiresUserAction field, and the high-urgency escalation condition. The model returns a JSON object. The service parses the result, removes any formatting artefacts that may be present, and validates each field before using it to select a routing action. The end-to-end processing loop is summarised in Algorithm 1.

Figure 7 expands the classification step of the polling service into its internal stages. The inputs are the incoming message (sender, subject and the first 400 characters of the body), the user's plain-English rules, and a small number of example phrasings for each rule category. These are assembled into a prompt consisting of a system message and a user message and sent to meta-llama/llama-3.1-8b-instant through the Groq API at a temperature of 0.1, in a single inference call per message. The structured JSON output is parsed and validated, and its fields are reduced to the outcome (i^*, g, c) together with the review flag h (the requiresUserAction field), which feed the routing decision $d(M)$. The figure shows that no task-specific fine-tuning is involved and complements the configuration reported in section 5.3.4.

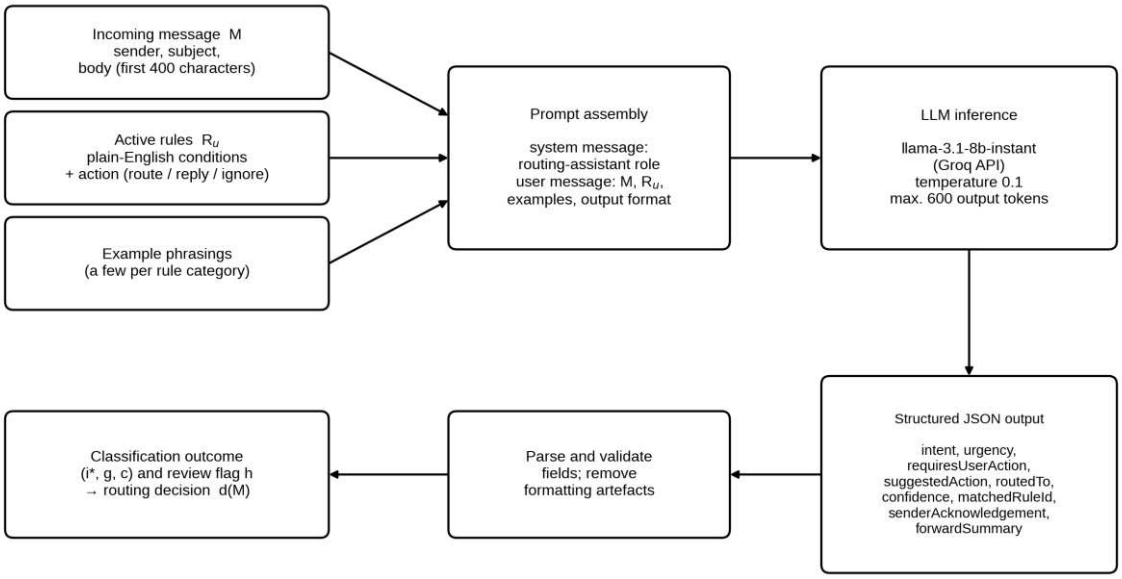


Fig. 7 Classification engine pipeline: prompt assembly, LLM inference, and output parsing and validation

Algorithm 1 Polling, classification, and routing loop

```
Input : connected account set A for user u, active rule set R_u,
        polling interval T_p
Output: routing action or human-review escalation, logged to Activity
loop every T_p:
    for each account a in A:
        for each new message M retrieved from a:
            f(M, R_u)                                   // LLM classification, section 3.3(i*, g, c, h) ←
            if i* ≠ ∅ and h = false and g ≠ high:
                log(M, i*, g, c, a_i*)                  // route, reply, or ignoreexecute a_i*
            else:
                escalate M to human-review queue
                log(M, i*, g, c, "escalated")
```

The distinction between M being routed automatically and being escalated is decided by a single, auditable check on the matched rule, the review flag and the urgency level, giving the platform one point of control over how conservatively it acts on ambiguous classifications.

Figure 8 presents the processing loop of Algorithm 1 as an activity diagram. Its purpose is to show how the platform operates once configured, without user involvement. After each polling interval T_p, the background service retrieves new messages from every connected Gmail and Slack account. For each unprocessed message it loads the user's active rules and requests a classification. If a rule is matched, the model does not flag the message for user action (h = false) and the urgency is not high, the corresponding action (forwarding, automated reply or ignoring) is executed; otherwise the message is placed in the human-review queue. Both paths end by writing an Activity record, after which the loop moves to the next message or the next polling cycle. The diagram therefore makes the abstain-and-escalate policy explicit: no message is acted on speculatively, and every outcome is auditable.

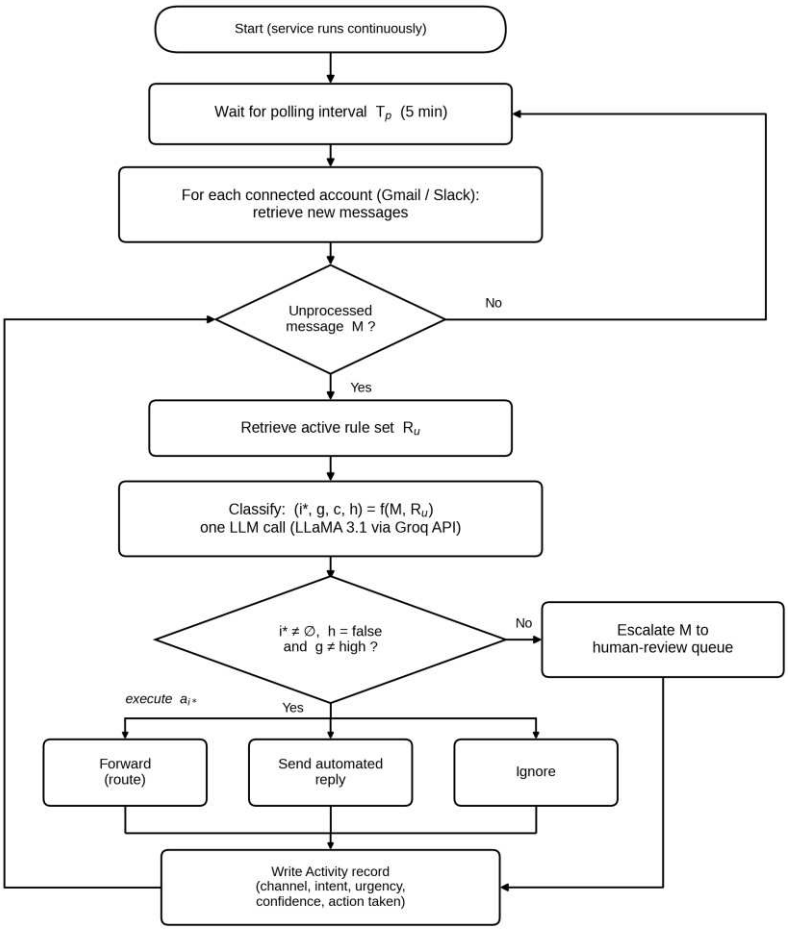


Fig. 8 Activity diagram of the polling, classification, and routing loop (Algorithm 1)

4.2.5 Dashboard and Activity Log

Next.js dashboard provides a consolidated view of channel connection status, active rules, and recent processing records. The activity log gives the user visibility of each automated decision, showing each processed message with the message classification, matched rule and resulting action. Importantly, the Activity schema (section 3.5.3) does not store message body content, only sender, subject, channel and classification metadata, so the processing history retained by the platform is by construction free of the underlying message text.

4.3 Development Approach

The system was built out iteratively over four development cycles: from architecture and data models, to authentication and channel integration, to classification engine and rule management, to the dashboard and end to end testing. This sequencing, more a practical dependency order (there is no meaningful way to test later modules without the modules that precede them) than a claim of methodological novelty, is reported here only as it explains the order in which the implementation is presented.

Figure 9 summarises how the platform was built and tested. Development proceeded through four iterative cycles: architecture and data models; authentication and channel integration; the classification engine and rule management; and the dashboard with end-to-end testing. Testing then took place at three levels: functional testing of individual components (34 tests; Table 5), integration testing of the full pipeline using a Gmail test account, and the preliminary classification feasibility check on 14 author-constructed messages (Tables 6 and 7). The order of the cycles follows the dependency order of the modules, so that each cycle builds on components already in place.

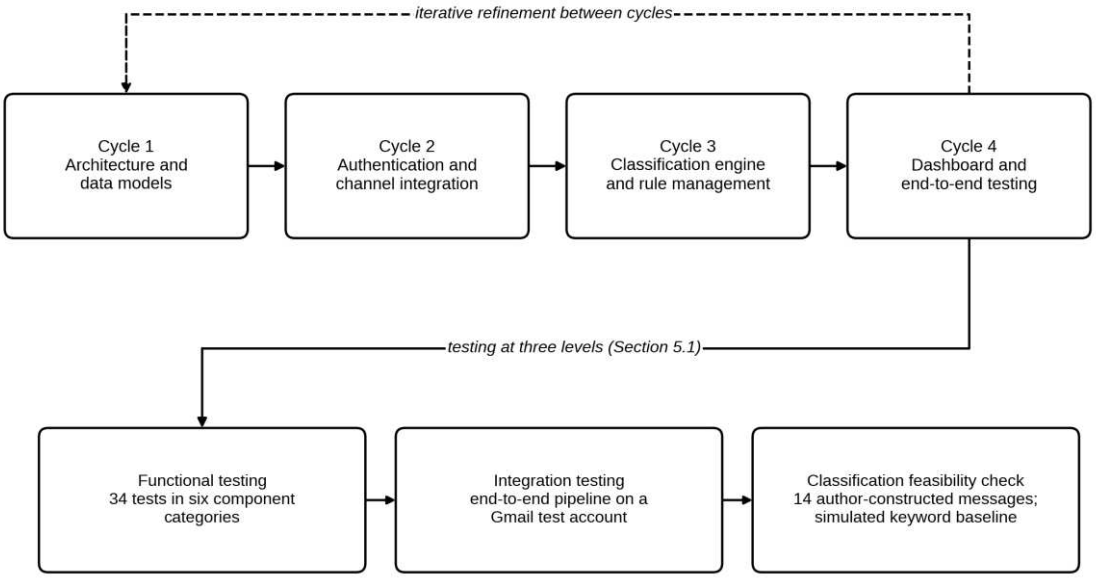


Fig. 9 Iterative development cycles and the three-level testing strategy

5 Testing and Evaluation

5.1 Testing Strategy

The platform was tested at three levels: functional testing of individual components, integration testing of the complete message processing pipeline and a preliminary feasibility check of the classification engine described in sections 3.3 and 4.2.4. All testing was performed using real messages sent to a connected Gmail test account and processed by the running platform, not synthetic API responses. The classification feasibility check in Section 5.3 used the Gmail test account only; classification of Slack messages was not evaluated.

5.2 Functional and Integration Testing

Functional testing verified that each system component was working correctly as stand-alone, including authentication, rule create, read, update and delete (CRUD) operations, Gmail and Slack integration, activity logging, and human-escalation handling. The results are summarised in Table 5. All 34 tests in six component categories passed.

Table 5 Functional test results

Test Category	Tests Conducted	Passed	Pass Rate
User authentication	5	5	100%
Rule management (CRUD)	8	8	100%
Gmail integration	6	6	100%
Slack integration	5	5	100%
Activity logging	6	6	100%
Human escalation	4	4	100%

Integration testing confirmed that the whole pipeline described in Algorithm 1 works as expected: messages sent from a test account were correctly detected by the polling service, classified, matched against active rules, acted on and logged. A representative example is an email received with the subject “I need help with my invoice from last month”, which was classified as billing-related, matched to the corresponding rule and forwarded to the configured destination address, with the outcome logged in the activity log. This confirms that the pipeline works end-to-end, and not just at the level of its individual components.

Figure 10 traces the invoice example above through the system as a sequence of interactions between the sender, the Gmail API, the polling service, the database and the LLM service. It shows the order of operations, which the

architecture figures do not: the polling service lists unread messages, receives the new message, fetches the user's active rules, requests a classification and receives the result as JSON. It then either forwards the message to the configured destination or flags it for human review, and finally records the outcome in the Activity collection. The diagram also shows that the delay before a message is first seen is the polling delay, which is bounded by T_p in Eq. (2).

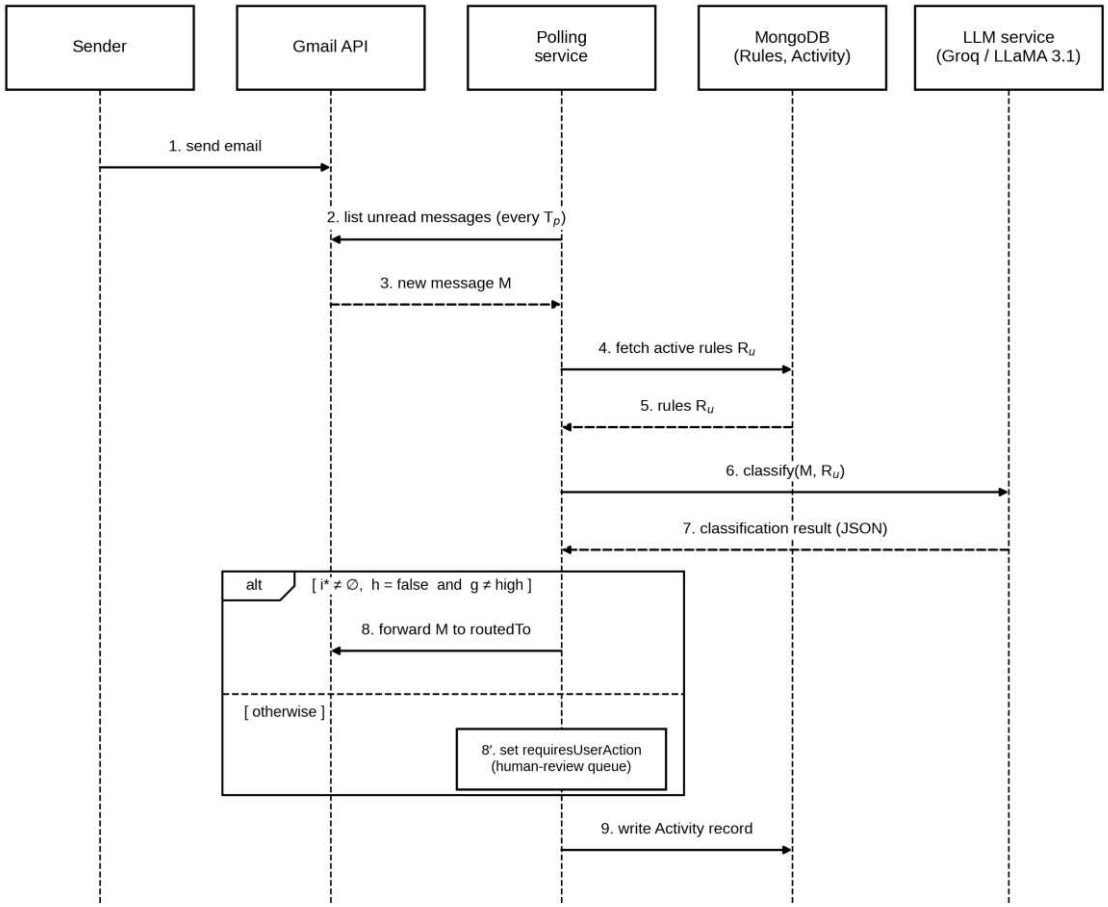


Fig. 10 Sequence diagram of end-to-end processing of one Gmail message (the invoice example)

5.3 Preliminary Classification Evaluation

5.3.1 Test Design

Fourteen test messages were created across four of the six rule schema supported intent categories: Billing, Scheduling, Partnership and General Inquiry; Complaints and Login Support were defined as valid rule categories but were not exercised in this test round. Messages were worded carefully to avoid the wording of its corresponding rule condition, so that the test would measure semantic matching rather than keyword overlap. For instance, the Billing rule used had a condition of "payment, invoice, billing or charges" and the test messages were "My card was charged twice this month" and "I need help with my invoice from last month" neither of which contain a word from the rule condition. The messages were written by the authors, and their expected categories were assigned by the authors without independent annotation, so no inter-annotator agreement can be reported. They are short, single-sentence texts and do not include signatures, quoted threads, forwarded chains, multiple topics, typographical errors, code-switching or Nigerian Pidgin, all of which real business email can contain. No separate development set was used, and the rules and prompt were not modified after the results were inspected. Correctness was judged by the authors by comparing the returned intent label with the intended rule, so that free-text labels such as "payment issue" were accepted as matches for the Billing rule.

A simple keyword-matching baseline was used for comparison with the LLM classifier. The baseline associated manually selected keywords with the four evaluated rule categories: Billing (`payment`, `invoice`, `billing`,

`charges`), Scheduling (`schedule`, `scheduled`, `meeting`, `call`), Partnership (`partner`, `partnership`, `collaboration`), and General Inquiry (`question`, `information`, `inquiry`). Matching was based on the presence of these terms in the message text. The baseline correctly matched 5 of the 14 evaluation messages (35.7%). The remaining messages were not matched to their intended categories, particularly where the messages expressed the underlying intent without using the corresponding rule terminology. The baseline was included as a simple lexical comparison rather than as a production routing method.

5.3.2 Results

Table 6 presents the results for each test message. Table 7 compares the classifier to a simulated keyword-matching baseline, intended to approximate the approach taken by Gmail filters and similar tools, applied to the same 14 messages.

Table 6 Classification results by test message

#	Message	Expected Category	Classified As	Correct
1	I haven't received my invoice yet	Billing	billing	Yes
2	My card was charged twice this month	Billing	payment issue	Yes
3	I need help with my invoice from last month	Billing	payment	Yes
4	I would like to book a call with your team	Scheduling	scheduling a call	Yes
5	Are you free for a meeting on Friday	Scheduling	scheduling	Yes
6	Please schedule me in for a demo	Scheduling	scheduling a demo	Yes
7	Can we find a time to speak this week	Scheduling	no match	No
8	Can we schedule a quick call for tomorrow	Scheduling	no match	No
9	I have a business proposal I would like to share	Partnership	business proposal	Yes
10	I have a quick question about your service	General Inquiry	general question	Yes
11	Can someone help me understand how this works	General Inquiry	general question	Yes
12	I would like some more information please	General Inquiry	general question	Yes
13	What are your opening hours	General Inquiry	general question	Yes
14	I need some help figuring something out	General Inquiry	general question	Yes

Overall accuracy was 12/14 (85.7%; 95% Wilson confidence interval (CI) 60.1–96.0%). Per-category accuracy was 3/3 (100%) for Billing, 3/5 (60%) for Scheduling, 1/1 (100%) for Partnership, and 5/5 (100%) for General Inquiry. With one to five messages per category, the per-category figures carry little evidential weight, and Complaints and Login Support were not tested.

Figure 11 shows classification accuracy for each intent category tested, with the dashed line marking overall accuracy (85.7%). Scheduling is the only category below 100%; the two unmatched messages both fall in it. With between one and five messages per category, these figures carry little evidential weight.

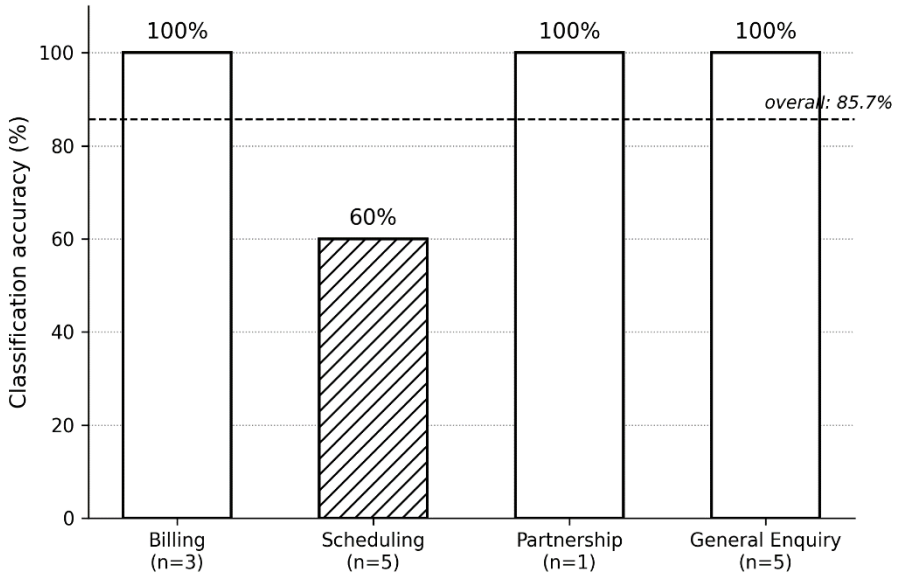


Fig. 11 Classification accuracy of the LLM classifier by intent category (number of test messages in parentheses); the dashed line shows overall accuracy (85.7%). Complaints and Login Support were not exercised

Table 7 Preliminary comparison of the classifier with a simulated keyword baseline (14 author-constructed messages)

Method	Correct Matches	Total Messages	Accuracy
Keyword baseline (simulated)	5	14	35.7% (95% CI 16.3–61.2%)
LLM classifier, example-augmented prompting (proposed system)	12	14	85.7% (95% CI 60.1–96.0%)
Difference	+7	—	+50 percentage points (not statistically established)

Figure 12 gives a visual summary of the comparison reported in Table 7. It shows the share of the 14 test messages matched to the intended rule by the LLM classifier (12 of 14; 85.7%) and by the simulated keyword baseline (5 of 14; 35.7%). The error bars show 95% Wilson intervals, which are wide and overlap slightly, so the figure should be read as an illustration of behaviour on this small, author-constructed set and not as evidence of comparative accuracy.

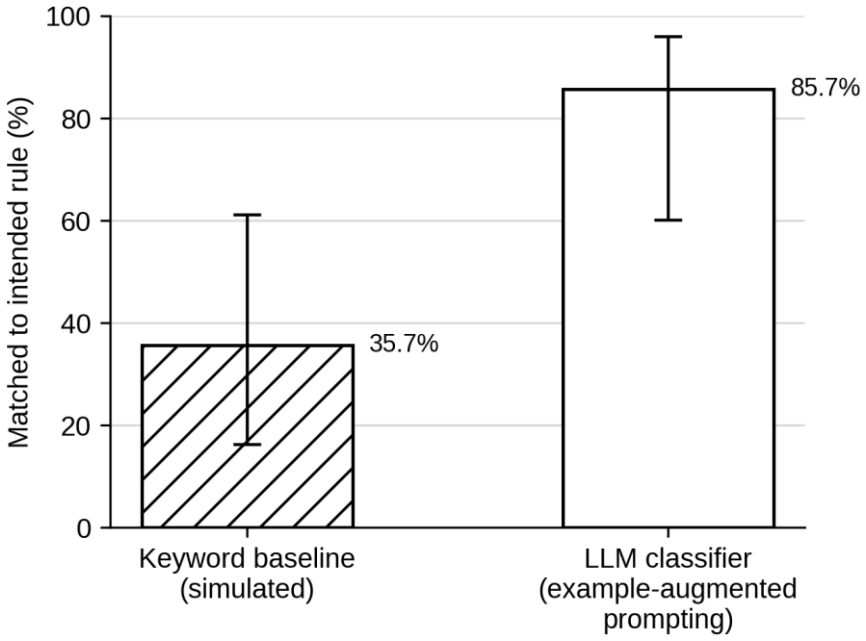


Fig. 12 Classification accuracy of the LLM classifier (example-augmented prompting) versus a simulated keyword-matching baseline on the same 14-message author-constructed test set. Bars show point estimates (85.7% and 35.7%); error bars show 95% Wilson intervals (60.1–96.0% and 16.3–61.2%)

A finer grained reading of Table 6 is useful in addition to overall accuracy. In both error cases, the system did not misroute the message to an incorrect rule, but returned no match, so that the messages were escalated to human review. Both error cases returned no match ($i^* = \emptyset$) and were therefore escalated to human review under the decision rule in Section 3.3. Twelve of the 14 messages were routed to the correct category; the other two were escalated rather than routed, which is not counted as a correct routing. This test run gives 12 true positives (TP), 2 false negatives (FN) and 0 false positives (FP), where a correct match is a TP, a failure to match an existing correct rule is an FN, and a match to an incorrect rule is an FP. Precision, recall and their harmonic mean, the F1 score (F1), are therefore:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{12}{12} = 1.00$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{12}{14} \approx 0.857$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \approx 0.923$$

Figure 13 presents precision, recall and F1 computed from TP = 12, FP = 0 and FN = 2. Precision is 1.00 because the two unmatched messages were escalated rather than misrouted and are therefore counted as false negatives, not false positives; recall is 0.857 and F1 is 0.923.

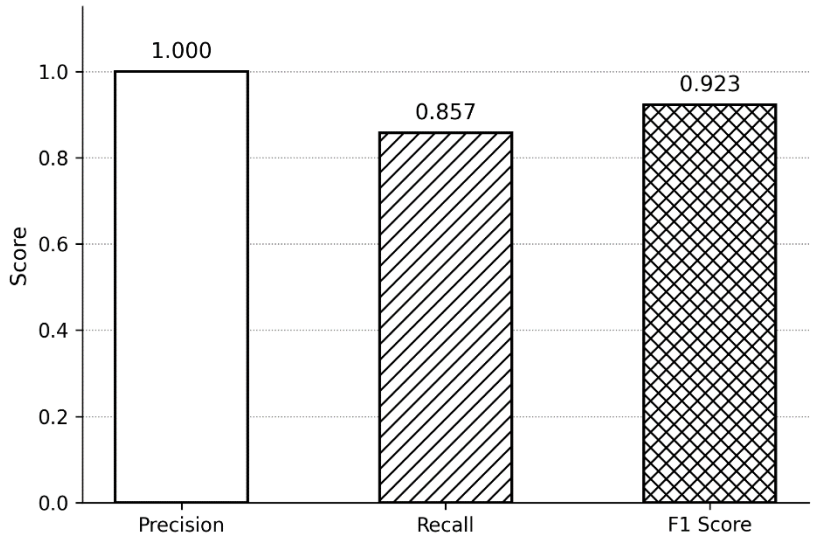


Fig. 13 Precision, recall and F1 score of the LLM classifier on the 14-message test set (TP = 12, FP = 0, FN = 2)

The distinction matters operationally: the system's errors in this test manifested entirely as reduced recall (missed matches), not reduced precision (incorrect matches). The precision of 1.00 follows from the way abstentions are scored: escalated messages are counted as false negatives rather than as errors, so precision must be read together with the abstention rate, which here was 2/14 (14.3%; coverage 85.7%). With 14 messages and a single configuration, these figures say little about the precision–coverage trade-off; a sweep over a confidence threshold θ is left to future work (Section 7.2).

5.3.3 Error Analysis

Both misclassified messages were from the Scheduling category: “Can we find a time to speak this week” and “Can we schedule a quick call for tomorrow”. Both implicitly express scheduling intent, with phrasing less tied to the rule condition (“booking a meeting”, “scheduling a call”, “arranging a demo”) than Scheduling messages correctly classified. This means that classification accuracy is sensitive to both the specificity of the incoming message and of the rule condition itself; a rule condition that enumerates a wider range of phrasings — e.g. including “find a time” or “quick call” as example expressions — might capture these two cases, but this was not tested, and the rule was not modified after these results were inspected. Tuning rules on test errors would make the test set no longer held out, so future evaluations must keep development and test data separate. This dependence on rule quality is a known property of prompt-based classification (Brown et al. 2020), not a bug of this particular implementation, and motivates the rule-quality guidance discussed in section 7.2.

5.3.4 Reproducibility

The classification configuration is reported in full for reproducibility: model meta-llama/llama-3.1-8b-instant via the Groq API; temperature 0.1; maximum output length 600 tokens; example-augmented prompting with semantically varied example phrasings included per rule category and no task-specific fine-tuning; and a prompt composed of a system message establishing the routing-assistant role and a user message containing the sender, subject, message body (truncated to 400 characters), and active rule set. The model output was requested as JSON with the fields intent, urgency, requiresUserAction, suggestedAction, senderAcknowledgement, forwardSummary, routedTo, confidence, and matchedRuleId. All reported results come from a single evaluation run, and the execution date was not recorded. Hosted-model outputs can vary between runs and over time, and repeated runs were not performed. The source code is not publicly available; the full prompt, rule definitions, keyword baseline, test messages and scoring scripts may be provided by the authors on reasonable request (see Data and Code Availability).

6 Discussion

The aim of the evaluation reported here was modest: to check whether an LLM classifier using example-augmented prompting can match differently worded messages to plain-English rules that a keyword filter would be expected to miss. The results are consistent with this on a small probe set (12/14 versus 5/14 for a simulated keyword baseline; Tables 6 and 7), but they do not establish that the approach routes messages more accurately than keyword matching. The 95% intervals for the two accuracies (60.1–96.0% and 16.3–61.2%) overlap slightly (60.1–61.2%); the messages were written by the authors specifically to avoid the rule wording, which favours the semantic approach by construction; the baseline is simple and was not compared with supervised, embedding-based or natural language inference (NLI)-based alternatives or with other LLMs; and only one 8B model was run, in one configuration and one run. The size of the gap should therefore be read as a property of the test design and not as an estimate of real-world performance.

The precision–recall distinction in sub-section 5.3.2 is, in the authors' view, the more informative of the two evaluation results. A classifier that achieves 85.7% accuracy by occasionally misrouting messages to the wrong destination would raise legitimate concerns about deployment in a business setting, where an incorrectly routed message can cause more disruption than a delayed one. The present system's errors instead took the form of appropriately conservative non-action: both misclassified messages were deferred to human review rather than acted on incorrectly. This behaviour is consistent with the abstain-and-escalate decision rule d(M) in sub-section 3.3, which the authors regard as a design strength. With only two errors in 14 messages, however, the test cannot show how often the classifier would misroute rather than abstain; establishing that requires a larger evaluation with a precision–coverage analysis across confidence thresholds.

The affordability and accessibility motivation for this work — a platform usable by resource-constrained Nigerian and African SMEs without dedicated technical staff — is reflected in the system's design (no-code plain-English rules, browser-based access, no fine-tuning or labelled-data requirement) rather than validated empirically here. This distinction is worth stating plainly: the present evaluation demonstrates technical feasibility and classification performance, not adoption, usability, or cost-effectiveness for the target user population. Sub-section 7.2 identifies this as the most important direction for follow-up work.

A further limitation concerns test coverage: the 14-message evaluation exercised four of the six intent categories the rule schema supports (Billing, Scheduling, Partnership, General Inquiry), leaving Complaints and Login Support unevaluated, and drew on a single test account rather than multiple users or organisations. The reported accuracy and F1 figures should be read as an initial, small-sample demonstration of the approach's viability rather than a statistically powered estimate of production accuracy.

The test data impose further limitations. The messages were constructed and labelled by the authors, were worded to avoid the rule conditions (which favours semantic over keyword matching by construction), and were short, clean, English-only texts; they do not reflect signatures, quoted threads, multi-topic messages, typographical errors, or local usage such as code-switching and Nigerian Pidgin. Slack messages were not part of the classification test; only one 8B-parameter model was evaluated, in a single configuration and a single run; and the keyword baseline was the only comparator. Stronger baselines (for example term frequency–inverse document frequency (TF-IDF) with logistic regression, embedding similarity, off-the-shelf zero-shot NLI classifiers, and larger LLMs) were not tested. The message body was truncated to 400 characters due to token cost the effect of this truncation on longer emails was not evaluated.

Several practical matters were not assessed. Per-message latency, cost, and API rate-limit behaviour were not systematically measured, although affordability is a central motivation. Per-message inference latency, end-to-end processing latency, API cost per message, and rate-limit behaviour were not systematically measured in this study. The implemented polling interval was five minutes, which provides an upper bound of five minutes on message-detection delay under the fixed-interval polling design, excluding subsequent classification and action-execution time. Classification requires sending message content to a third-party LLM API; although the platform's activity log does not store message bodies (Section 4.2.5), operators should assess this transfer under applicable data-protection rules, including the Nigeria Data Protection Act 2023 (Federal Republic of Nigeria 2023), for example regarding lawful basis, notice to correspondents, and cross-border transfer. Because message content is

sent to a third-party LLM API, privacy-preserving machine-learning techniques (Acheme et al. 2024) offer a possible route to limiting the exposure of message content in future versions of the platform. Finally, automated routing decisions carry governance risks: a misrouted or ignored complaint, for instance, may harm a customer. The escalation rule and audit log mitigate this, but organisations may wish to require human review for high-consequence categories such as complaints and to review the activity log periodically.

7 Conclusion and Future Work

7.1 Conclusion

This study presented the design, implementation, and preliminary evaluation of an intelligent message routing and triage platform that classifies incoming Gmail and Slack messages by semantic intent, using a pre-trained LLM accessed via the Groq API, and routes them according to plain-English rules defined by the end user. The platform integrates OAuth-based Gmail and Slack access, an LLM classification engine using example-augmented prompting, and a conservative escalation policy (no matching rule, model-flagged review, or high urgency) within a full-stack TypeScript architecture (section 3 and 4).

Functional and integration testing confirmed that all reviewed components and the end-to-end processing pipeline operate as designed (Table 5). The preliminary feasibility check matched 12 of 14 author-constructed messages to the intended rule (85.7%; 95% CI 60.1–96.0%), against 5 of 14 (35.7%; 16.3–61.2%) for a simulated keyword baseline, and the two remaining messages were escalated to human review rather than misrouted (precision of routed messages 1.00; abstention rate 14.3%). These results are consistent with the feasibility of semantic matching and of the abstain-and-escalate policy but, given the size and construction of the test set, they do not establish that the approach is more accurate than keyword matching or other baselines.

The broader contribution is a documented system design — combining an externally hosted LLM, OAuth-based communication APIs, and a plain-English, no-code rule interface — for embedding semantic classification into multi-channel business productivity tools, with particular relevance to small organisations that can neither justify the cost of enterprise automation platforms nor invest in the labelled-data and fine-tuning infrastructure that supervised or fine-tuned ticket-classification systems require (sub-sections 2.2–2.4); whether the design generalises beyond the small test reported here, and how it compares with stronger baselines, remains to be established.

7.2 Limitations and Future Work

Several directions follow from the limitations identified in section 6:

- *Empirical evaluation with SME users.* The most important next step is a study involving actual Nigerian and African SME users, including usability testing, cost–benefit comparison against existing alternatives, and measurement of routing accuracy across the diverse, naturally occurring message types such organisations handle. This would substantiate the affordability and accessibility claims that motivate the work, which the present study establishes by design rather than by user evidence. Beyond routing accuracy, future studies could relate triage performance to customer-experience outcomes in web-based retail, building on determinants of consumer experience identified in earlier work (Acheme and Nwankwo 2026).
- *Broader and larger-scale classification evaluation.* Future evaluation should extend to the two untested categories (Complaints, Login Support), use an independently labelled set of several hundred messages (with a reasonable number per category, for example 30–50 or more) that includes naturally occurring anonymised messages, a Slack subset, and local-language variation, with inter-annotator agreement reported. It should keep development and test data strictly separate, compare against stronger baselines (for example TF-IDF with logistic regression, embedding similarity, zero-shot NLI classifiers, and additional LLMs), repeat runs with the model version and date recorded, and report accuracy, macro-averaged precision, recall and F1, abstention rate and confidence intervals, together with a sensitivity analysis of the confidence threshold θ.

- *Real-time message processing.* The five-minute polling interval bounds latency at T_p (section 3.3) but is not instantaneous. Webhook-based delivery — Gmail Pub/Sub push notifications and the Slack Events API — would remove this bound where the underlying platform supports it.
- *Rule-quality guidance.* Since classification accuracy is sensitive to rule specificity (sub-section 5.3.3), a rule-quality advisor that suggests improvements based on observed low-confidence or escalated classifications would help non-technical users improve routing accuracy without requiring them to understand the underlying model.
- *Classification accuracy improvements.* Larger LLMs, a user feedback loop for correcting misclassifications, and ablations that separate the contribution of rule wording from that of the example phrasings (strictly zero-shot versus example-augmented prompting) are natural extensions.
- *Additional channels and languages.* Extending beyond Gmail and Slack to Microsoft Outlook, Microsoft Teams, and the WhatsApp Business API is supported by the platform's API-first architecture without fundamental redesign (section 3.1). Similarly, English-only classification limits applicability in linguistically diverse African markets; evaluating performance in French, Yoruba, Igbo, Hausa, and other widely used business languages is an important extension for the platform's stated target context.

References

Abramovich G (2019) If you think email is dead, think again. Adobe, 8 September 2019

Acheme ID, Nwankwo W (2026) Identifying key determinants for consumer experience in web-based retail stores using an improved fuzzy-Bayesian model. NIPES Journal of Science and Technology Research 8:185–196

Acheme ID, Olayinka A, Jegede A et al (2023) Investigating the most influential factors for customer satisfaction in online stores. In: 2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS), vol 1. IEEE, pp 1–4

Acheme S, Edegbe GN, Acheme ID (2024) Advances and challenges in privacy-preserving machine learning. In: 2024 IEEE SmartBlock4Africa. IEEE, pp 1–8

Brown TB, Mann B, Ryder N et al (2020) Language models are few-shot learners. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). https://doi.org/10.48550/arXiv.2005.14165

Chui M, Manyika J, Bughin J et al (2012) The social economy: unlocking value and productivity through social technologies. McKinsey Global Institute, New York

Federal Republic of Nigeria (2023) Nigeria Data Protection Act 2023. Federal Government of Nigeria, Abuja

Galitsky B (2020) Tree kernel learning for text classification into classes of intent. US Patent 10,839,161 B2, Oracle International Corporation

Google (n.d.) Manage Gmail filters. Gmail API documentation. Google, Mountain View. https://developers.google.com/gmail/api/guides/filter_settings. Accessed 18 September 2026

GSMA (2022) The mobile economy: sub-Saharan Africa 2022. GSMA, London

Jackson TW, Dawson R, Wilson D (2003) Understanding email interaction increases organizational productivity. Commun ACM 46:80–84. https://doi.org/10.1145/859670.859673

Liu F, He X, Zhang T et al (2025) TickIt: leveraging large language models for automated ticket escalation. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE '25), Trondheim. https://doi.org/10.1145/3696630.3728558

Liu H, Zhang X, Fan L et al (2019) Reconstructing capsule networks for zero-shot intent classification. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Association for Computational Linguistics, Hong Kong, pp 4799–4809. https://doi.org/10.18653/v1/D19-1486

Liu Z, Benge C, Jiang S (2023) Ticket-BERT: labeling incident management tickets with language models. arXiv preprint arXiv:2307.00108. https://doi.org/10.48550/arXiv.2307.00108

Make Technologies (2026) Make pricing. https://www.make.com/en/pricing. Accessed 18 September 2026

Marcuzzo M, Zangari A, Schiavinato M et al (2022) A multi-level approach for hierarchical ticket classification. In: Proceedings of the Eighth Workshop on Noisy User-generated Text (W-NUT 2022). Association for Computational Linguistics, Gyeongju, pp 201–214. https://aclanthology.org/2022.wnut-1.22

Microsoft (n.d.) Power Automate documentation. Microsoft, Redmond. https://learn.microsoft.com/power-automate/. Accessed 18 September 2026

Min S, Lyu X, Holtzman A et al (2022) Rethinking the role of demonstrations: what makes in-context learning work? In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP 2022). Association for Computational Linguistics, Abu Dhabi, pp 11048–11064. https://doi.org/10.18653/v1/2022.emnlp-main.759

Montgomery L, Damian D (2017) What do support analysts know about their customers? On the study and prediction of support ticket escalations in large software organizations. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp 362–371. https://doi.org/10.1109/RE.2017.61

Radicati Group (2023) Email statistics report, 2023–2027. Radicati Group, Palo Alto

Shi W, Mariano VY (2025) Enhanced intention recognition model based on text classification. IEEE Access 13:40226–40236. https://doi.org/10.1109/ACCESS.2025.3547017

Slack Technologies (2023) Slack usage statistics. Slack, San Francisco

Slack Technologies (n.d.) Guide to Slack Workflow Builder. Slack Help Center. https://slack.com/help/articles/360035692513-Guide-to-Slack-Workflow-Builder. Accessed 18 September 2026

Statista (2023) Number of Gmail users worldwide. Statista, New York

Wei J, Bosma M, Zhao VY et al (2022) Finetuned language models are zero-shot learners. In: International Conference on Learning Representations (ICLR 2022). https://doi.org/10.48550/arXiv.2109.01652

Xian Y, Lampert CH, Schiele B et al (2019) Zero-shot learning — a comprehensive evaluation of the good, the bad and the ugly. IEEE Trans Pattern Anal Mach Intell 41:2251–2265. https://doi.org/10.1109/TPAMI.2018.2857768

Zapier Inc. (2026) Zapier pricing. https://zapier.com/pricing. Accessed 18 September 2026

Zeldes N, Sward D, Louchheim S (2007) Infomania: why we can't afford to ignore it any longer. First Monday 12(8). https://doi.org/10.5210/fm.v12i8.1973

Zoho Corporation (2024) The state of productivity and collaboration in Nigeria. Zoho, Chennai

Statements and Declarations

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

Author Contributions

All authors contributed to the study conception and design

Ethics Approval and Consent to Participant

Not applicable: this study did not involve human participants or animals; the test messages were written by the authors.

Data Availability

The 14 test messages are listed in Table 6. The application source code is maintained in a private repository and is not publicly available. The full system prompt, rule definitions, keyword baseline, per-message classifier and baseline outputs, and evaluation scripts are not currently deposited in a public repository or DOI-linked archive; they may be made available by the authors upon reasonable request, subject to applicable privacy, security, and third-party service restrictions.