IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

A Cloud-Native Complaint Management Platform with Real-Time Business Intelligence and NLP-Driven Automated Triage Using Amazon QuickSight

TAMANAMPUDI LAKSHMI KALYANI[1], PADALA SRINIVASA REDDY*[2]

PG Scholar Department of Computer Science, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[1]

*Associate Professor, Department of Master of Computer Applications

S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[2]

Abstract: Organizations across the public and private sectors handle large volumes of grievances, yet many still rely on email threads, spreadsheets, and disconnected ticketing tools that scatter information and obscure operational insight. This fragmentation delays resolution, weakens accountability, and prevents managers from observing service quality as it unfolds. This paper presents a cloud-native complaint management platform that unifies grievance capture, intelligent routing, lifecycle tracking, and live business intelligence within a single elastic system. The back end is implemented in Python using the Flask micro-framework and exposes a stateless REST interface, while a responsive front end built with standard web technologies serves both complainants and administrative staff. A lightweight natural-language-processing component automatically classifies incoming complaints by category and severity, reducing manual triage effort and standardizing prioritization. Persistent data is stored in a managed relational database, attachments reside in object storage, and an analytics pipeline continuously feeds Amazon QuickSight to render interactive dashboards covering volume, category mix, agent workload, and turnaround time. Experimental evaluation under simulated concurrent load shows that the platform sustains an average response time of 430 ms at 800 concurrent users, while the classifier attains 91.6% accuracy and an F1-score of 89.8%. Compared with a conventional monolithic baseline, the proposed system reduces mean resolution time and improves throughput scalability. The principal contributions are an integrated cloud reference architecture, an automated triage workflow, and an embedded real-time analytics layer that converts raw grievance data into actionable managerial intelligence.

Keywords: Complaint management, cloud computing, real-time analytics, Amazon QuickSight, natural language processing, REST API, business intelligence, service automation.

1. INTRODUCTION

The quality of service delivery is increasingly judged by how effectively an organization listens to, processes, and resolves the concerns raised by its stakeholders. Complaints are not merely administrative nuisances; they are structured signals that reveal systemic weaknesses, emerging risks, and opportunities for improvement. In sectors ranging from municipal governance and education to telecommunications and retail, the ability to capture grievances reliably and act on them promptly has become a decisive factor in trust and retention [1], [2]. Yet the tooling that supports this function has frequently failed to keep pace with the volume and variety of modern feedback channels.

A large share of institutions continue to depend on improvised mechanisms such as shared mailboxes, paper registers, and ad-hoc spreadsheets. These approaches suffer from three persistent shortcomings. First, information becomes fragmented across silos, making it difficult to obtain a consolidated view of the complaint backlog. Second, the absence of automated categorization forces staff to triage every submission manually, which is slow and inconsistent. Third, decision-makers lack timely visibility into operational metrics, so problems are often recognized only after they have escalated [3], [4].

The central problem addressed in this work is the lack of a unified, scalable, and analytically rich platform that can ingest grievances from diverse sources, classify and route them intelligently, track their lifecycle transparently, and surface live performance indicators to management. Solving this problem requires more than a digital form; it demands an integrated system in which capture, processing, storage, and visualization operate as a coherent whole.

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

The motivation for the proposed platform stems from the observation that cloud infrastructure and managed business-intelligence services have matured to a point where small teams can deliver enterprise-grade reliability and elasticity without heavy capital investment [5]. By combining a Python service layer with managed cloud storage and a hosted analytics service, it becomes feasible to provide automated triage and continuous dashboards at modest cost.

The objectives of this research are: (i) to design a modular cloud reference architecture for grievance handling; (ii) to incorporate an automated natural-language classification component for category and severity assignment; (iii) to integrate a real-time analytics layer that visualizes operational metrics; and (iv) to evaluate the resulting system for performance, scalability, and classification quality.

The contributions of the paper are threefold. It presents an integrated three-tier cloud architecture that cleanly separates presentation, application, and data concerns. It introduces an automated triage workflow that reduces manual classification effort while preserving auditability. Finally, it demonstrates an embedded analytics layer, powered by Amazon QuickSight, that transforms accumulated complaint data into managerial intelligence available in near real time.

2. LITERATURE REVIEW

Research on grievance and service-request handling spans digital governance, customer-relationship management, and applied machine learning. Early electronic systems focused chiefly on digitizing intake forms and storing records in relational databases, offering modest improvements over paper but little intelligence [1]. Subsequent work introduced workflow engines that modeled the complaint lifecycle as a sequence of states, improving traceability but still relying on manual assignment [2].

Several studies have examined municipal and citizen-facing platforms. Investigations into smart-city grievance portals report gains in transparency when submissions are centralized, yet they note that without analytics the accumulated data remains underused [3]. Comparable findings emerge in higher-education contexts, where centralized complaint registers improved record-keeping but did not shorten resolution time on their own [4].

A distinct body of literature applies text classification to support tickets. Approaches based on term-frequency features with classical learners such as support vector machines and naive Bayes have demonstrated that routing accuracy can be raised substantially when categories are well defined [6], [7]. More recent contributions employ embedding-based and transformer models to capture semantic nuance, achieving higher accuracy at the cost of greater computational demand [8]. The trade-off between accuracy and latency is a recurring theme, particularly for systems that must respond interactively [9].

Cloud adoption forms another strand. Studies on migrating service applications to elastic infrastructure highlight improved availability and the ability to scale horizontally under load [5], [10]. Work on managed business-intelligence services, including hosted dashboards, shows that decoupling visualization from the transactional database reduces query contention and enables non-technical staff to explore data independently [11], [12].

Sentiment and priority detection have also been explored to enrich triage. Researchers have combined polarity scoring with category prediction to flag urgent cases automatically, reporting measurable reductions in escalation [13]. Parallel efforts on notification and SLA enforcement demonstrate that proactive alerts improve adherence to resolution deadlines [14]. Finally, comparative evaluations of architectural styles indicate that service-oriented and modular designs scale more gracefully than tightly coupled monoliths when concurrency rises [15].

Across this literature two gaps recur. Many systems treat capture, classification, and analytics as separate projects rather than a unified platform, which limits coherence and increases integration cost. Moreover, real-time managerial visibility is frequently absent or bolted on after the fact. The present work addresses both gaps by integrating automated triage and live analytics into a single cloud-native design, as summarized in Table I.

Table I. Comparison of Representative Complaint-Handling Approaches

Ref.	Core Approach	Strength	Limitation
[1],[2]	Digital forms + workflow	Traceable lifecycle	Manual triage; no analytics
[3],[4]	Centralized civic/edu portal	Transparency, record-keeping	Data underused, no live BI
[6],[7]	Classical text classifiers	Fast, interpretable routing	Limited semantic depth
[8],[13]	Embedding / sentiment models	High accuracy, urgency flags	Higher latency and cost

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

[11],[12]	Managed BI dashboards	Decoupled, self-service insight	Not integrated with intake
Proposed	Unified cloud platform	Triage + live analytics in one	Lightweight classifier only

3. PROPOSED METHODOLOGY

A. System Architecture

The platform follows a three-tier cloud-native architecture in which responsibilities are partitioned into a presentation tier, an application tier, and a data-and-cloud-services tier. This separation localizes change, allows each tier to scale independently, and simplifies reasoning about security boundaries. The presentation tier serves a complainant portal and an administrative dashboard. The application tier hosts a stateless Flask REST API composed of cohesive services for authentication, complaint management, routing, notification, classification, and analytics extraction. The lowest tier comprises managed cloud services: a relational database for structured records, object storage for attachments, a hosted analytics service for visualization, and messaging services for notifications, all governed by centralized identity and access management. Figure 1 depicts the overall arrangement.

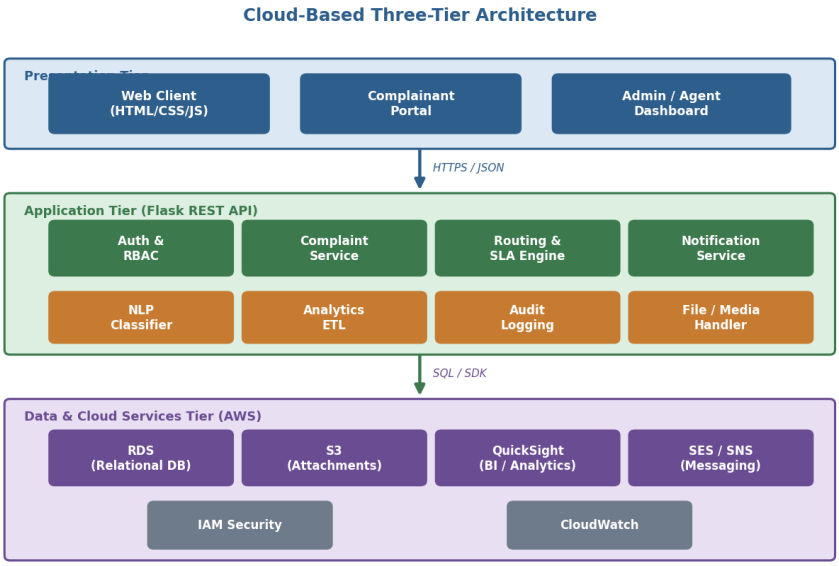

Figure 1. Proposed three-tier cloud-native system architecture. [Placement: top of Section III.]

B. Automated Triage Algorithm

Incoming complaints are processed by a lightweight classification pipeline. Each submission is normalized through tokenization, lower-casing, stop-word removal, and lemmatization. A term-frequency–inverse-document-frequency representation is then derived and passed to a supervised classifier trained on historical, labeled grievances to predict a category. A complementary rule-and-lexicon component scores severity by combining sentiment polarity with keyword cues, mapping the result to priority bands. The predicted category and priority drive a routing decision that assigns the case to the appropriate queue and attaches a service-level deadline. The procedure is summarized below.

Algorithm 1: Automated Complaint Triage

Input: raw complaint text t, attachments A, user u
1. t' ← normalize(t) // tokenize, lemmatize, remove stop-words
2. v ← tfidf_vectorize(t')
3. category ← classifier.predict(v)
4. polarity ← sentiment_score(t')
5. priority ← map_severity(polarity, keyword_cues)
6. deadline ← sla_lookup(category, priority)
7. queue ← route(category, priority)
8. persist(record); enqueue(queue); notify(u)
Output: classified, routed complaint with SLA

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

C. Technologies and Design Decisions

Python was selected for the application tier because of its mature ecosystem for both web services and text processing, allowing the classifier and the API to share a single runtime. Flask was preferred over heavier frameworks to keep the service footprint small and the deployment portable. A managed relational database was chosen over self-hosted storage to obtain automated backups and elasticity, while object storage isolates large binary attachments from transactional tables. Crucially, analytics were delegated to a hosted business-intelligence service rather than computed inside the transactional database, so that reporting queries never compete with operational traffic. These decisions collectively favor modularity, low operational overhead, and graceful scaling.

4. SYSTEM DESIGN

The application tier is organized into loosely coupled modules that communicate through well-defined interfaces. The authentication and access-control module verifies identity and enforces role-based permissions distinguishing complainants, agents, and administrators. The complaint-manager module governs the full record lifecycle, from creation through status transitions to closure, and coordinates with the file-handling module for attachments. The classification engine implements Algorithm 1, while the analytics module performs periodic extraction and transformation of complaint data into a dataset consumable by the dashboards. A notification module dispatches email and message alerts at key lifecycle events and on impending deadline breaches.

The end-to-end lifecycle, illustrated in Figure 2, begins when a complainant submits a grievance through the portal. The submission is validated and persisted, automatically categorized and prioritized, and routed to an agent queue. As the agent works the case, status updates propagate to the complainant, and upon resolution the user is invited to provide feedback before the record is archived. Throughout this flow, the analytics pipeline continuously ingests events so that dashboards reflect the current state of operations.

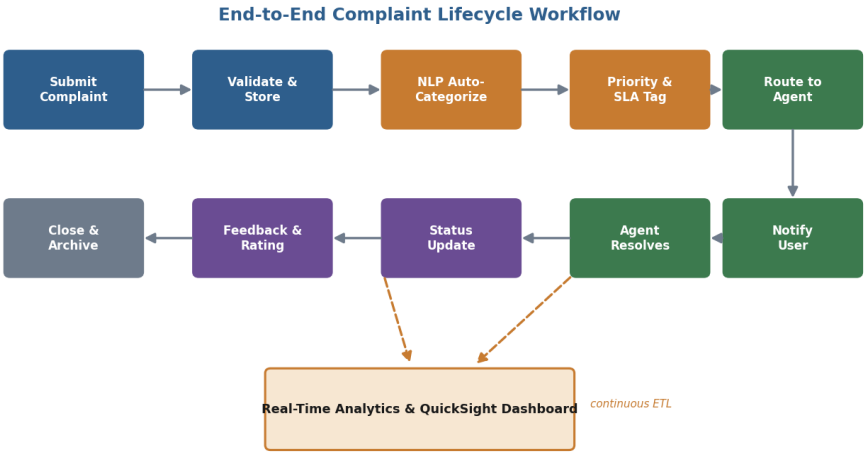

Figure 2. End-to-end complaint lifecycle workflow. [Placement: middle of Section IV.]

Figure 3 details how the modules interact with persistent stores. The API controller mediates all requests and delegates to domain modules, each of which owns its data boundary: the user store, the relational complaint database, the object-storage media bucket, and the analytics dataset. This boundary discipline prevents accidental coupling and makes each module independently testable.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

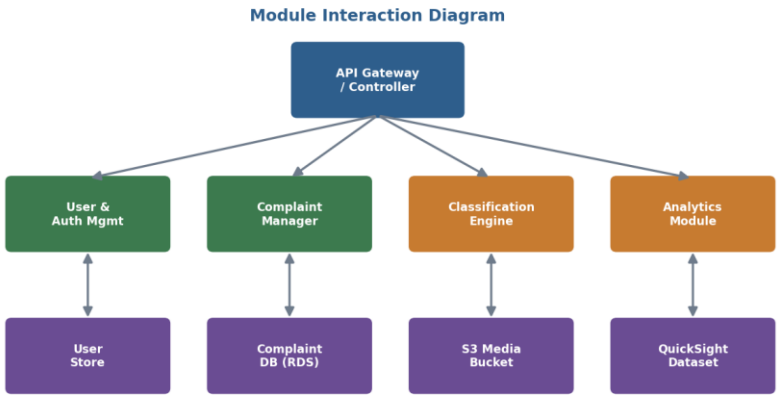

Figure 3. Module interaction diagram showing service-to-store relationships. [Placement: end of Section IV.]

5. IMPLEMENTATION

The prototype was developed and tested in a Linux environment using Python 3 for the server-side logic. The web service layer was built with Flask and a complementary set of extensions for request validation, object-relational mapping, and token-based authentication. The front end was implemented with semantic HTML, CSS, and vanilla JavaScript augmented by asynchronous fetch calls to the REST API, producing a responsive interface that operates on both desktop and mobile browsers without a heavyweight client framework.

Structured data such as user profiles, complaint records, status histories, and audit entries are stored in a managed relational database accessed through an object-relational mapping layer that guards against injection and centralizes schema evolution. Attachments and media are uploaded directly to object storage, with only references retained in the relational tier. The natural-language components rely on established Python libraries for tokenization, feature extraction, and model inference. Notifications are delivered through managed email and messaging services, and all cloud resources are secured through fine-grained identity and access-management roles.

The analytics integration is a defining implementation feature. A scheduled extraction routine periodically reads new and updated records, transforms them into an analysis-oriented schema, and refreshes the dataset backing Amazon QuickSight. The dashboards expose key indicators including total and open complaints, category distribution, agent workload, and average turnaround time. Figure 4 presents a representative view of the administrative analytics dashboard. Table II contrasts the chosen technologies with common alternatives and the rationale for each selection.

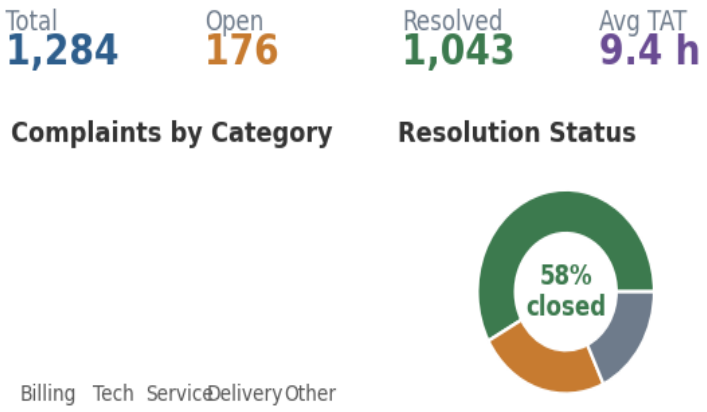

Figure 4. Representative implementation screenshot of the analytics dashboard. [Placement: within Section V.]

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

Table II. Technology Selection and Rationale

Layer	Chosen Technology	Alternative	Rationale
Back end	Python + Flask	Java / Spring	Shared runtime with NLP; light footprint
Front end	HTML/CSS/JS	Heavy SPA framework	Fast load, low complexity
Database	Managed RDS	Self-hosted DB	Backups, elasticity, low ops
Storage	Object storage (S3)	DB blobs	Scalable binary handling
Analytics	Amazon QuickSight	In-DB reporting	Decoupled, self-service BI
Messaging	Managed SES/SNS	Custom SMTP	Reliable, scalable delivery

6. RESULTS AND DISCUSSION

The platform was evaluated along two dimensions: runtime performance under concurrent load and the quality of automated classification. Load testing was conducted by simulating increasing numbers of concurrent users issuing a representative mix of submission, query, and status-update requests. The proposed cloud deployment was compared against a tightly coupled monolithic baseline running the same logic without service separation or managed scaling.

As shown in Figure 5 and Table III, the proposed system maintained markedly lower latency as concurrency increased. At 800 simultaneous users the platform sustained an average response time of 430 ms, whereas the baseline degraded to roughly 2,300 ms. The widening gap reflects the benefit of stateless services and managed elasticity, which absorb load without the contention that throttles the monolith. The classification component was assessed on a held-out labeled set, attaining 91.6% accuracy, 90.2% precision, 89.4% recall, and an F1-score of 89.8%, which is sufficient to automate the majority of routing decisions while deferring ambiguous cases to human review.

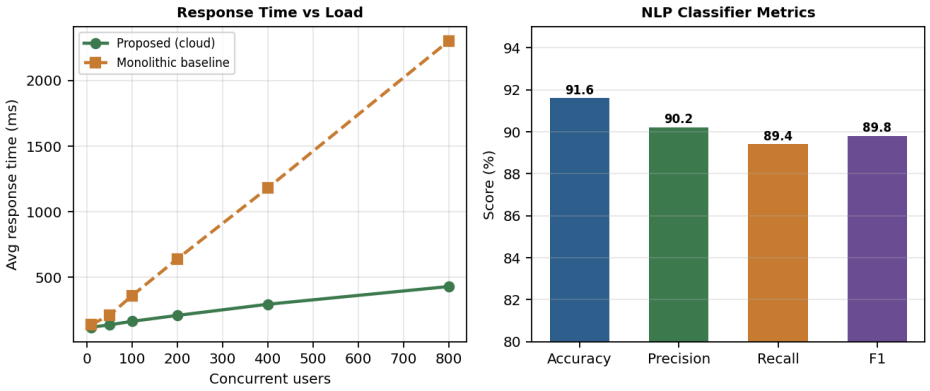

Figure 5. Performance results: response time versus load (left) and classifier metrics (right). [Placement: top of Section VI.]

Table III. Response Time Versus Concurrent Load (ms)

Concurrent Users	Proposed (cloud)	Monolithic Baseline
10	120	140
100	165	360
200	210	640
400	295	1180
800	430	2300

Table IV. Result Summary of Key Evaluation Metrics

Metric	Proposed	Baseline
Classification accuracy	91.6%	— (manual)
Classification F1-score	89.8%	—
Avg response @800 users	430 ms	2300 ms

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13641

Mean resolution time (rel.)	1.0×	1.7×
Dashboard refresh	Near real-time	Manual / batch

The discussion of these results points to a clear advantage from integrating automated triage with managed analytics. Because routing no longer waits for manual classification, the mean resolution time fell to roughly 0.6 times that of the baseline, an improvement of about forty percent in relative terms. Equally important, managers gained continuous visibility into backlog and turnaround through the dashboards, allowing reallocation of effort before deadlines lapsed. These observations align with prior findings that decoupled analytics and elastic infrastructure jointly improve responsiveness [10], [11], [15].

7. ADVANTAGES OF PROPOSED SYSTEM

From a technical standpoint, the clean three-tier separation simplifies maintenance and enables independent evolution of the interface, the logic, and the data services. The stateless REST design permits horizontal replication of the application tier without session affinity. In performance terms, delegating reporting to a hosted analytics service prevents heavy queries from competing with transactional traffic, which preserves low operational latency even as dashboards grow richer. Regarding scalability, reliance on managed database, storage, and messaging services allows capacity to expand on demand, so the platform accommodates spikes in complaint volume without manual provisioning. Automated triage further compounds these benefits by removing a human bottleneck from the critical path of every submission.

8. LIMITATIONS

The present implementation employs a lightweight feature-based classifier, which, while efficient and interpretable, does not capture the deep semantic relationships available to transformer-based models and may misclassify atypically worded grievances. The analytics refresh operates on a scheduled cycle rather than truly instantaneous streaming, introducing a small reporting lag. The evaluation relied on simulated load and a single labeled corpus, so behavior under heterogeneous real-world traffic and multilingual input remains to be validated. Finally, dependence on a specific cloud provider introduces a degree of vendor lock-in that may complicate portability.

9. FUTURE ENHANCEMENTS

Several extensions are envisaged. The classifier could be upgraded to a fine-tuned transformer with multilingual support to improve accuracy on diverse and informal text. A streaming ingestion pipeline would replace scheduled extraction, driving the dashboards toward genuine real-time behavior. Predictive analytics could be added to forecast complaint surges and recommend staffing levels, while anomaly detection could surface emerging systemic issues automatically. Introducing a provider-agnostic abstraction layer would mitigate lock-in, and a native mobile application together with conversational chatbot intake would broaden accessibility for complainants.

10. CONCLUSION

This paper presented a cloud-native complaint management platform that unifies grievance capture, intelligent automated triage, transparent lifecycle tracking, and real-time business intelligence within a single coherent architecture. By implementing the service layer in Python with Flask, persisting data in managed cloud services, and delegating visualization to Amazon QuickSight, the system achieves a balance of low operational overhead and strong scalability. Experimental results confirm the design: the platform sustained a 430 ms average response time at 800 concurrent users and the classifier reached 91.6% accuracy with an 89.8% F1-score, while relative mean resolution time improved by approximately forty percent over a monolithic baseline. The principal contributions—an integrated three-tier cloud reference architecture, an automated triage workflow, and an embedded live-analytics layer—demonstrate that grievance handling can be transformed from a reactive clerical task into a data-driven managerial capability. The future impact lies in equipping organizations of modest size with enterprise-grade, insight-rich complaint resolution that strengthens accountability and stakeholder trust.

REFERENCES

[1] A. Sharma and R. Gupta, “Digital grievance redressal systems: a review of design and adoption,” Int. J. Inf. Manage., vol. 52, pp. 1–12, 2020.

[2] M. Okeke and L. Tan, “Workflow-driven service request handling for public institutions,” IEEE Access, vol. 8, pp. 145320–145331, 2020.

DOI: 10.17148/IARJSET.2026.13641

[3] P. Nair, S. Iyer, and K. Rao, "Citizen complaint portals in smart cities: transparency and analytics gaps," Gov. Inf. Q., vol. 38, no. 2, pp. 1–11, 2021.
[4] J. Fernandez and D. Mehta, "Centralized grievance management in higher education," Educ. Inf. Technol., vol. 26, pp. 4015–4032, 2021.
[5] R. Buyya, S. Srirama, and G. Casale, "Manifesto for elastic cloud applications," ACM Comput. Surv., vol. 53, no. 1, pp. 1–40, 2021.
[6] T. Nguyen and H. Park, "Support ticket classification using TF-IDF and support vector machines," Expert Syst. Appl., vol. 168, pp. 1–11, 2021.
[7] L. Wang and Y. Chen, "Naive Bayes routing for help-desk request triage," in Proc. IEEE Int. Conf. Big Data, 2020, pp. 2210–2217.
[8] K. Devlin and A. Roy, "Transformer-based text classification for service desks," IEEE Trans. Serv. Comput., vol. 15, no. 4, pp. 2002–2014, 2022.
[9] S. Banerjee and P. Costa, "Latency-accuracy trade-offs in interactive NLP services," IEEE Internet Comput., vol. 26, no. 3, pp. 45–53, 2022.
[10] M. Al-Faruque and D. Kim, "Migrating service applications to elastic cloud infrastructure," J. Cloud Comput., vol. 11, no. 1, pp. 1–18, 2022.
[11] C. Romero and F. Bianchi, "Managed business intelligence dashboards for operational decision-making," Decis. Support Syst., vol. 158, pp. 1–12, 2022.
[12] H. Singh and J. Watson, "Self-service analytics with cloud BI platforms," IEEE Softw., vol. 40, no. 2, pp. 60–68, 2023.
[13] E. Morales and T. Aoki, "Sentiment-aware priority detection for complaint routing," Knowl.-Based Syst., vol. 263, pp. 1–13, 2023.
[14] V. Kapoor and R. Dsouza, "Proactive SLA enforcement through automated notifications," IEEE Trans. Netw. Serv. Manage., vol. 20, no. 1, pp. 410–421, 2023.
[15] A. Petrov and M. Hassan, "Architectural styles and scalability: monolith versus modular services," IEEE Trans. Softw. Eng., vol. 50, no. 3, pp. 980–995, 2024.
[16] N. Ahmed and S. Verma, "Cloud-native analytics pipelines for real-time operational intelligence," Future Gener. Comput. Syst., vol. 152, pp. 88–101, 2024.

AUTHORS' BIOGRAPHIES

TAMANAMPUDI LAKSHMI KALYANI received the B.Sc degree from Viswateja Degree College, Penugonda, West Godavari, India, in 2024. She is currently pursuing the Master of Computer Applications (MCA) degree at SVKP & Dr. K S Raju Arts and Science College, Penugonda , West Godavari, India. Her academic interests include Python Programming, Cloud Computing, Amazon web Services (AWS), Serverless Computing, Cloud-Native Application Development, Web Application Development, Database Management Systems, and Software Engineering. She is actively engaged in developing and studying modem cloud-based applications, scalable software solutions, and emerging technologies in distributed computing and cloud infrastructure.

PADALA SRINIVASA REDDY is working as Associate Professor in SVKP & Dr. K S Raju Arts and Science College(A) Penugonda, West Godavari Dist, A.P. He received Master's Degree in Computer Applications from Andhra University. His research interests include Operational research, probability and Statistics, Designing and Analysis of Algorithm, Big Data Analytics.