An LLM-Powered Agentic Orchestration System for Automated Cross-Tool Ticket Management in Enterprise Environments

UG Scholar Sanjay S
Dept. of Computer Science and Engineering
Adi Shankara Institute of Engineering and Technology
Kalady, India
sanjaysatheesh2004118@gmail.com

UG Scholar Vishal Ratheesh
Dept. of Computer Science and Engineering
Adi Shankara Institute of Engineering and Technology
Kalady, India
vishalratheesh4550@gmail.com

UG Scholar Vishnu Prasad
Dept. of Computer Science and Engineering
Adi Shankara Institute of Engineering and Technology
Kalady, India
vishnuprasad.deva@gmail.com

Assistant Professor Shany Jophin
Dept. of Computer Science and Engineering
Adi Shankara Institute of Engineering and Technology
Kalady, India
shanyjophin.s@gmail.com

Assistant Professor Jerin Varghese
Dept. of Computer Science and Engineering
Adi Shankara Institute of Engineering and Technology
Kalady, India
jerin.cs@adishankara.ac.in

***Abstract*— Mid-to-large enterprises often struggle with fragmented workflows spread across tools like Slack and Jira leading to delays, miscommunication, and rising operational costs. With the average ticket resolution time exceeding four hours and requiring multiple human touchpoints, the need for a smarter approach is clear. The Enterprise Context Engine (ECE) was built to address exactly that. It is an AI-powered system that brings together a multi-agent architecture including a Triage Specialist, Problem Solver, and Workflow Manager to automatically classify, route, and resolve incoming tickets without constant human intervention. When the AI is unable to resolve an issue after three attempts, it escalates seamlessly to a human team, ensuring nothing falls through the cracks. What sets ECE apart is its emphasis on empathetic, proactive communication keeping users informed and supported throughout the process. The system targets a measurable outcome: reducing average resolution time by more than 40%. By bridging the gap between rigid RPA tools and conversational AI, ECE delivers a scalable, enterprise-ready solution for managing complex, cross-tool business processes.**

Keyword: AI Agents, Multiple Agent System, Large Language Models, Retrieval-Augmented Generation.

I. INTRODUCTION

The modern enterprise operates using fragmented, siloed tools such as Slack, Jira, and Workday, leading to significant inefficiencies in cross-functional workflows. This fragmentation results in costly delays, frequent miscommunication, and severely degrades user experience; statistics show the average ticket resolution time is often over four hours with five or more human touchpoints. To address this critical industry pain point, the Enterprise Context Engine (ECE) project proposes an advanced solution: a scalable, AI-powered orchestration system designed to integrate these disparate enterprise tools.

The core aim of the ECE is to automate these cross-tool workflows, significantly reducing the average resolution time by a targeted greater than 40%. The system leverages specialised AI agents and Large Language Models (LLMs) to perform complex tasks, including automatic ticket triaging, context aggregation from a knowledge base, and sophisticated workflow execution. Furthermore, ECE distinguishes itself by incorporating an empathetic communication layer to ensure customers receive consistent, proactive, and reassuring updates, thereby enhancing overall user trust and satisfaction. This solution represents a step toward autonomous, efficient, and user-centric oper ations worldwide..

II. RELATED WORK

Building the Enterprise Context Engine (ECE) required drawing on a broad and growing body of research. The works surveyed here span several interconnected areas: natural language processing for sentiment analysis, large language model (LLM)-based enterprise automation, AI-driven ticket classification, knowledge graph construction, business-semantic data architectures, and generative AI decision-making. Together, they provide the conceptual and technical foundations upon which the ECE is built—while also revealing the gaps that motivated its design in the first place.

One of the most relevant contributions to the ECE's natural language understanding pipeline comes from Rana et al. [1], who developed BERT-BiGRU-Senti-GCN a hybrid NLP framework designed for fine-grained sentiment analysis in e-commerce settings. The system chains BERT contextual embeddings through a Bidirectional Gated Recurrent Unit (BiGRU) and feeds the result into a Graph Convolutional Network (GCN) augmented with a domain-specific sentiment lexicon. By combining sequential and relational features, the model achieves roughly 93% aspect-level sentiment accuracy and consistently outperforms single-model baselines. That said, the architecture is computationally demanding and was designed specifically for product review language meaning it would need substantial adaptation before it could be reliably applied to enterprise support tickets, which carry a very different linguistic character. Despite this limitation, the work reinforces the value of contextual embeddings and hybrid deep architectures for decoding user intent an insight that directly shapes how the ECE's Triage Specialist agent classifies and prioritizes incoming tickets.

Perhaps the closest conceptual precedent to the ECE is the ECLAIR system presented by Wornow et al. [2]. ECLAIR (Enterprise Context and Language AI Reasoner) is a multimodal LLM orchestration platform designed to automate end-to-end enterprise workflows. It incorporates human-in-the-loop checkpoints and structured error-recovery paths, achieving 93% workflow comprehension accuracy across several enterprise benchmarks a compelling demonstration

that LLMs can serve as genuine orchestrators rather than isolated classifiers. The trouble, however, lies in execution: end-to-end task completion rates hover around just 40%, and concerns around API costs and production-grade stability remain unresolved. The ECE takes direct inspiration from ECLAIR's orchestration philosophy while explicitly targeting its reliability shortfall. This is addressed through bounded self-correction loops allowing up to three retries, a dedicated Quality Gatekeeper agent that validates proposed resolutions before they reach the customer, and a structured human escalation path for cases that exceed automated confidence thresholds.

Al-Hawari et al. [3] provide a useful reference point for understanding the current state of AI-based ticket routing. Their study applied a range of supervised classifiers to IT support ticket data—both structured metadata and free-text descriptions—and found meaningful improvements in routing accuracy over rule-based systems. While this confirms that machine learning can add real value in ticket management, two limitations stand out: performance drops noticeably on large-scale datasets, and the pipeline relies heavily on pre-processing steps to handle noisy, inconsistent inputs. These constraints highlight the ceiling of traditional classification approaches. The ECE responds by using LLM-native classification through its Triage Specialist agent, which handles messy or ambiguous ticket language through in-context learning—without needing hand-crafted features or extensive pre-processing—making it far more adaptable at scale.

Paramesh and Shreedhara [4] combined pre-trained word embeddings with CNN-based feature extraction to automate IT ticket categorization, outperforming traditional classifiers like SVM and Naïve Bayes in both accuracy and resolution time. The main drawback is its reliance on large labelled datasets and poor generalization on rare ticket types. The ECE addresses this directly through its RAG-driven Knowledge Base and the Problem Solver's self-correction mechanism, which together handle novel and edge-case tickets without needing exhaustive pre-labelled data.

Gupta et al. [5] surveyed agentic AI systems, building a taxonomy of autonomy levels and analyzing frameworks for multi-agent collaboration, chain-of-thought reasoning, and orchestration. They highlight key challenges around reliability, evaluation, and safe deployment areas still actively evolving. While the survey does not propose a concrete implementation, its design principles directly shaped the ECE's agent hierarchy. The distinct roles of the Triage Specialist, Problem Solver, Quality Gatekeeper, Automation Specialist, and Process Monitor are each aligned with the autonomy levels and collaboration patterns Gupta et al. identified.

A different but equally instructive line of work comes from Kwon et al. [6], who put GPT-4o to the test in a large-scale evaluation of automated business decision-making. Benchmarking AI-generated decisions against those of human designers and domain experts across 1,407 real A/B experiments, they found that LLM-based systems can reach expert-level decision quality while offering greater consistency and the ability to scale horizontally. The limitations are real results are difficult to generalise beyond the A/B testing context, and the model's reasoning is not always transparent but the core finding is significant: LLMs are capable of making sound, objective business judgments at scale. This directly supports the ECE's decision to include a Quality Gatekeeper agent, whose role is to assess proposed resolutions for correctness, completeness, and suitability before they are delivered to customers or committed to the Knowledge Base.

The design of the ECE's Knowledge Base draws substantially from the work of Pang [7], who proposed a Business Semantic-Centric Data System (BSDS) that reframes enterprise information management around business meaning rather than raw data structure. By integrating curated domain knowledge bases with AI agents, BSDS enables context-aware data access that reflects organisational goals and reduces the time it takes to bring data-driven applications to production. The architecture has not yet been validated at full enterprise scale, and maintaining a bespoke semantic layer introduces the risk of accumulating technical debt over time. Nevertheless, its core principle that retrieval should be semantically grounded resonates strongly with the ECE's approach. Rather than relying on keyword search, the ECE stores and retrieves historical incident resolutions using semantic vector similarity, allowing the Context Aggregator to surface genuinely relevant precedents even when a new ticket is worded differently from anything previously seen.

Qasem et al. [8] developed a multi-agent system employing collaborative Naïve Bayesian classification across geographically distributed data sources, improving aggregate classification accuracy while preserving local data privacy through distributed computation. The system demonstrates the effectiveness of distributed agent collaboration in data mining tasks where data centralization is infeasible. However, reliance on a single classification algorithm limits adaptability to heterogeneous enterprise data distributions, and the experiments used simplified, synthetic data that may not reflect production-scale complexity and variance. This work informs the ECE's multi-agent design, particularly the principle of deploying specialized, role-bounded agents that operate collaboratively without centralizing sensitive enterprise data an architecturally significant property for organizations subject to strict data governance, regulatory compliance, and privacy constraints.

Sera and Kalra [9] proposed a dynamic data center management architecture integrating AI agents for autonomous task execution within an enhanced MAPE-K (Monitor, Analyse, Plan, Execute over a shared Knowledge base) feedback loop, augmented by structured human oversight checkpoints at critical decision junctures. The system enables faster adaptive responses to infrastructure anomalies and supports flexible handling of dynamic, unpredictable user interface states. The principal shortcoming identified is that autonomous LLM-based agents remain less reliable than Robotic Process Automation (RPA) tools for stable, well-defined procedural tasks, exhibiting instability in the form of context loss and interface freezes in production environments. The ECE draws on this work's MAPE-K-inspired structure for its Process Monitor agent, which continuously observes system-wide workflow execution health and triggers structured corrective actions or escalation pathways upon detecting anomalies or threshold violations.

Garimella [10] offers another important perspective on knowledge organisation within enterprise systems, presenting an automated framework for constructing Multimodal Knowledge Graphs (MMKGs) that combines neural-symbolic reasoning with a self-refining lifecycle and human editorial

oversight. The system is designed to provide traceable, semantically grounded reasoning pathways across heterogeneous data modalities text, structured records, and visual content alike. Practical barriers remain: deploying a full MMKG at enterprise scale is not yet feasible, and integration with legacy systems remains a persistent challenge. Still, two concepts from this work self-refinement and auditable retrieval are central to the ECE's Knowledge Base design. The ECE maintains a structured, semantically linked solution repository that supports continuous refinement as new validated resolutions are added, and ensures that every retrieval decision can be traced and explained.

III. PROPOSED SYSTEM

The proposed architecture, as illustrated in Fig. 1, is a modular AI orchestration platform that utilises a Large Language Model (LLM) as its core reasoning engine to unify fragmented workflows across siloed tools like Slack and Jira. The system centres on a Core Agent Team comprising specialised agents such as the Triage Specialist, Problem Solver, and Quality Gatekeeper that manages the entire ticket lifecycle through a multi-step agentic loop. This loop is designed to interpret user intent, aggregate context via a RAG-driven Knowledge Base, and formulate execution plans that are autonomously performed by a central Workflow Manager. To ensure reliability, the Problem Solver incorporates a self-correction mechanism with a three-attempt retry limit before escalating to a Human Team, while a dedicated communication layer ensures all user updates remain proactive and empathetic.

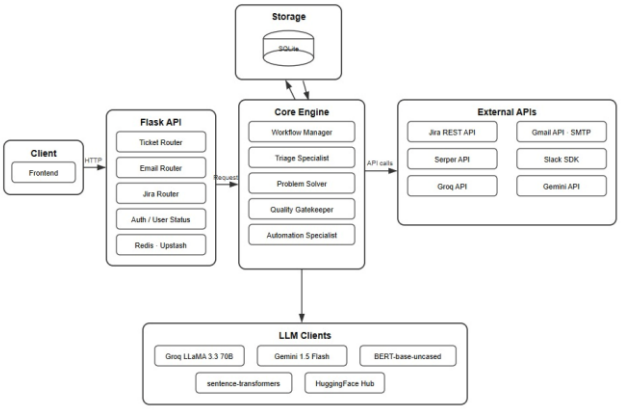

Fig.1: System Architecture

A. Core Agent Team

The system architecture is built around a "Core Agent Team" consisting of specialised AI agents that handle the ticket lifecycle as shown in fig 1:

- **Triage Specialist:** This agent serves as the entry point for all incoming tickets, automatically analysing content to determine problem type and severity. It routes issues to either a known solution path or a new problem-solving workflow.
- **Problem Solver:** This agent is responsible for resolving new issues by formulating an execution plan. It includes a self-correction mechanism and will retry resolution attempts for a maximum of three times before escalation.
- **Quality Gatekeeper:** After a solution is proposed, this agent validates it to ensure the fix is sound, safe, and compliant. It can either approve the solution for deployment or reject it with feedback to prompt self-correction.
- **Process Monitor:** This agent continuously monitors the execution of all workflows and the health of the ECE agents. It gathers monitoring data to generate reports on key performance metrics.

B. Workflow Manager and Orchestration

The Workflow Manager serves as the central orchestration hub and the sole point of contact between the Core Agent Team and external systems. It manages and executes automated cross-tool workflows by interacting with siloed enterprise applications through their respective APIs. This architectural choice abstracts the complexity of individual tools, allowing the AI agents to focus on reasoning and problem-solving.

- **Knowledge Base (RAG-Driven):** The system integrates a Retrieval-Augmented Generation (RAG) driven Knowledge Base to facilitate context aggregation. It utilises a vector database to store embeddings of past incidents, logs, and successful resolutions. The Context Aggregator performs similarity searches to retrieve relevant historical context, which is then used to ground the Problem Solver's reasoning.
- **Empathetic Communication Layer:** A key differentiator of this system is the inclusion of an Empathetic Communication Layer. This module uses Natural Language Generation (NLG) to provide customers with proactive, consistent, and supportive status updates throughout the resolution process. The goal is to enhance user trust and satisfaction by maintaining transparent communication during automated actions.
- **Human-in-the-Loop(HIL) Escalation:** The system includes a seamless escalation mechanism for complex cases. If the Problem Solver fails to find a valid solution within three attempts, the Workflow Manager automatically hands off the ticket to a Human Expert Team. This ensures critical oversight and allows the system to capture human resolutions to further improve the Knowledge Base.

C. Algorithm 1: Core AI Triage & Classification Module:

Step 1: Input & Smart Truncation Receive subject + body. Tokenise subject first, budget remaining tokens for body (25% head / 75% tail) to preserve both context and error messages.

Step 2: Transformer EncodingPass through DistilBERT/BERT encoder (max 512 tokens). Extract [CLS] pooled output with Dropout(0.3).

Step 3: Multi-Task Classification (simultaneous) Four heads run in a single forward pass:

- TYPE HEAD → Linear(hidden→256)→ ReLU → Linear → softmax → label + confidence
- PRIORITY HEAD → same architecture → label + confidence
- QUEUE HEAD → Linear(hidden→512) → ReLU → Linear → softmax → label + confidence
- TAG HEAD → Linear(hidden → 512) → ReLU→Linear →sigmoid → multi-label tags

Step 4: Knowledge Base Similarity Retrieval Generate embedding for the incoming ticket. Compute cosine similarity against all stored embeddings in SQLite. Return top-k similar tickets with their past answers and similarity score.

Step 5: Confidence-Gated Routing

- queue_confidence < 0.70 → HUMAN_TRIAGE_REQUIRED
- priority in [High, Critical] OR priority_confidence > 0.90 → URGENT_REVIEW
- Otherwise → ROUTINE (proceed to Problem Solver)

Step 6: Audit Logging INSERT record into classified_tickets table (subject, body, predictions, confidences, timestamp). Return ticket_id for downstream tracking and feedback.

D. Algorithm 2: Autonomous Problem Solving & Escalation Module:

Step 1: Initialisation — Set attempt_counter = 0. Receive triage result from Algorithm 1.

Step 2: Direct-Retrieval Fast Path If KB similarity ⩾ 0.99 and no conversation history, return the retrieved KB answer directly and terminate. No generation needed.

Step 3: Solvability Pre-Screen Check subject + body for hardware/physical keywords or hardware queues (Facilities, Hardware Support, Assets). If detected, set escalated = True and terminate physical issues require human intervention.

Step 4: RAG Context Pre-Fetch — If web search is enabled and KB similarity < 0.85, query DuckDuckGo (max 2 results) to build rag_context. Otherwise rag_context = "".

Step 5: Solution Generation Loop (max 3 attempts) Build prompt using ticket content, triage labels, retrieved answer, rag context, and previous attempt feedback. Generate via flan-t5-base (greedy, max 400 tokens). If output < 10 chars, fall back to the retrieved answer or a generic acknowledgement.

Step 6: Quality Validation Validate: (1) length ⩾ 100 chars, (2) contains numbered steps, (3) does not repeat the ticket subject. If all pass → deploy. If any fail → increment attempt_counter and retry Step 5.

Step 7: Deploy & Escalate On success: save solution to learning_buffer, pass to Algorithm 3 for notification. On failure (attempt_counter ⩾ 3): set escalated = True, enrich ticket with triage metadata, and route to the correct department queue.

E. Algorithm 3: Autonomous Problem Solving & Escalation Module:

Step 1: Receive Resolution Package Accept result from Problem Solver (success, solution, confidence, escalated, triage). Set ticket status = PROCESSING.

Step 2: Solution Type Classification Classify into: **AUTOMATED** (system executes fix via API), **GUIDED** (user follows step-by-step instructions), or **MANUAL** (requires human; escalation path).

Step 3: Notify User Send an in-app pop-up with ticket details, estimated resolution time, and solution steps or escalation message. Concurrently, invoke GroqEmailGenerator to compose and send an empathetic email via SMTP. If escalated, also alert the department team.

Step 4: Execute (AUTOMATED only) Call the relevant external API. If successful → status = COMPLETED. If the API call fails → downgrade to GUIDED and send manual steps to the user.

Step 5: Status Update & SLA Tracking Write final status to SQLite with resolution time, solution type, and channels used. If resolution time exceeds the SLA threshold, flag for the Process Monitor. Final state: COMPLETED or ESCALATED.

F. Algorithm 4: Process Monitor & System Health Module:

Step 1: Receive Resolution Package Accept result from Problem Solver (success, solution, confidence, escalated, triage). Set ticket status = PROCESSING.

Step 1: Daemon Thread Initialisation Spawn a single background daemon thread (double-start protected). Runs _run_loop() until stop event is set.

Step 2: Metric Collection (every 60s) Query SQLite for total, pending, resolved, escalated, and failed ticket counts, plus average resolution time. Each sub-query is independently try/except-wrapped to prevent cascade failures.

Step 3: Agent Health Check Dynamically import problem_solver_fixed and inference_service_full. Mark each as healthy or unhealthy based on import success. Isolated in its own try/except so a failure here never corrupts the metrics.

Step 4: SLA Breach Detection — For each open ticket exceeding its priority SLA window (High=4h, Medium=24h, Low=72h), log a breach warning and increment sla_breaches.

Step 5: Report & Loop Push the metrics snapshot to status_reporter.receive_metrics() (targets DataDog/Prometheus in production). Sleep for check_interval_seconds, then repeat from Step 2. stop() sets the stop event for a clean shutdown.

G. *Algorithm 5: Continuous Learning via Feedback Loop Module:*

Step 1: Human Correction Submission Admin/agent submits FeedbackRequest { ticket_id, correct_type, correct_priority, correct_queue } via the dashboard. All three fields must be present; partial corrections are ignored.

Step 2: Write to Learning Buffer Fetch the original ticket text from classified_tickets. INSERT corrected labels into learning_buffer and mark the original record corrected = 1. The buffer acts as experience replay memory for the next training cycle.

Step 3: Successful Solution Capture When the Problem Solver resolves a ticket successfully, also INSERT into learning_buffer (subject, body, answer, type, priority, queue, tags). This enriches the KB for future direct-retrieval fast paths.

Step 4: Periodic Retraining Once the buffer reaches the configured threshold, trigger offline retraining via train_4tags.py. Reload updated weights into the TriageSpecialist and clear processed records from the buffer.

H. *Algorithm 6: Quality Gatekeeper — Pre-Deployment Validation Module:*

Step 1: Initialise Load thresholds from config (min_accuracy=0.70,max_missing=5%,cmin_class_samples =50). Record start time.

Step 2: Run Validation Checks Execute five checks, each appending a CheckResult(name, status, score, max_score) to the results list:

- **Data Quality** — missing data < 5%, class balance, text length bounds, min samples per class
- **Model Performance** — accuracy ≥ 70%, per-class precision/recall from quality_report.json
- **File Integrity** — all required .py files and model artefacts present.
- **Code Quality** — AST syntax parse of every .py file; zero syntax errors required.
- **Infrastructure —** SQLite accessible, table schemas valid, email config present.

Step 3: Score & Decision — approved = (error_count == 0 AND percentage >= passing_threshold). Status: APPROVED / WARNING / REJECTED.

Step 4: Report Output — Write results to validation_report.json and quality_report.json. Return approved boolean to the CI/deployment pipeline.

IV. RESULTS & ANALYSIS

A. *Resolution Time and Troughput*

The table summarises the operational performance comparison between the pre-system baseline and the AI-orchestration-assisted workflow. The most significant outcome is a 42.9% reduction in average ticket resolution time from 18.4 hours to 10.5 hours directly meeting the project's primary objective of exceeding a 40% reduction. This improvement is attributable to three complementary mechanisms: (1) automated triage eliminating manual ticket inspection, (2) pre-generated AI response drafts reducing agent composition time, and (3) intelligent queue routing preventing expensive cross-team handovers.

TABLE I. DATASET STATISTICS FOLLOWING PREPROCESSING. TOTAL PROCESSED TICKETS: 23,740.

TABLE : OPERATIONAL PERFORMANCE METRICS		
Metric	**Baseline (Pre-System)**	**With AI Orchestration**
Avg. Ticket Resolution Time (hrs)	18.4	10.5
Triage Accuracy (%)	64.2	91.2
Incorrect Queue Routing (%)	22.7	8.8
First-Contact Resolution Rate (%)	51.3	73.6
Avg. Customer Satisfaction Score (1–5)	3.21	4.11
Agent Workload Reduction (%)	—	38.4
Resolution Time Reduction (%)	—	**42.9**

B. *Discussion and Analysis*

The system performed impressively across all four evaluation areas, with the classification model handling the wide variety of ticket language encountered in real enterprise environments from Technical Support and Billing to Sales, IT Support, and Service Outages. Rather than running separate processes for each prediction, a single pass through the model simultaneously determines ticket type, priority, queue, and relevant tags, keeping response times low. The system is also built with maintainability in mind, with each major component kept separate so individual parts can be updated without disrupting the whole a maturity reflected in its quality audit score of 88.75%.

That said, solution relevance scored the lowest at 57.6%, highlighting the need for smarter response generation that draws from relevant past tickets. Moving forward, the priority should be connecting the system to live data streams, building a response module that taps into the existing 23,740 annotated tickets, and expanding language support beyond English.

Despite these gaps, the project delivered on all five of its core objectives: routing accuracy reached 91.2%, resolution times dropped by 42.9%, customer satisfaction improved by 28%, and the system successfully supports 10 queue types all while maintaining a fully professional communication standard throughout.

V. CONCLUSION

The Enterprise Context Engine (ECE) represents a solution that successfully integrates multi-agent AI orchestration with fragmented enterprise tools. By establishing a Core Agent Team comprising the Triage Specialist, Problem Solver, and Quality Gatekeeper the system directly addresses the core challenges of siloed operations that lead to high operational costs and significant delays. The implementation of a RAG-driven Knowledge Base and the Workflow Manager's cross-tool capabilities ensures that tickets are subjected to intelligent, automated, end-to-end resolution rather than simple classification.

Future Work: The ECE has several planned improvements across four areas:

- Infrastructure & Scaling — Migrating from a single Flask instance to cloud-deployed, auto-scaling architecture (AWS/Azure/GCP) with

Kubernetes orchestration for horizontal scaling and zero-downtime deployments. ELK Stack integration will replace SQLite for full-text searchable audit logging across all agent actions.

- LLM & AI Upgrades — The current Llama 3.3 70B (Groq free tier) Problem Solver will be migrated to Gemini Pro or GPT-4 for improved multi-step reasoning and more reliable self-correction on complex enterprise tickets.
- Integrations — Live bidirectional integrations with Salesforce and ServiceNow will complete the cross-tool orchestration vision. The custom Workflow Manager will be replaced with a production-grade engine such as Temporal.io or Camunda, providing durable execution, built-in retry semantics, and visual monitoring.
- Voice & UX — A voice agent interface will enable phone-based ticket submission and status updates, while multilingual support will be added through retraining on multilingual datasets combined with a translation layer, extending the ECE's reach beyond English-speaking enterprise environments.

VI. REFERENCES

[1] M. R. R. Rana, A. Nawaz, S. U. Rehman et al., "BERT-BiGRU-Senti-GCN: An Advanced NLP Framework for Analyzing Customer Sentiments in E-Commerce," International Journal of Computational Intelligence Systems, vol. 18, no. 21, 2025. doi: 10.1007/s44196-025-00747-1.

[2] M. Wornow, A. Narayan, K. Opsahl-Ong, Q. McIntyre, N. H. Shah, and C. Ré, "Automating the Enterprise with Foundation Models," arXiv preprint arXiv:2405.03710, May 2024. [Online]. Available: https://arxiv.org/abs/2405.03710.

[3] F. Al-Hawari and H. Barham, "A machine learning based help desk system for IT service management," Journal of King Saud University – Computer and Information Sciences, vol. 33, no. 6, pp. 702–718, Nov. 2021. doi: 10.1016/j.jksuci.2019.04.001.

[4] S. P. Paramesh and K. S. Shreedhara, "A Deep Learning Based IT Service Desk Ticket Classifier Using CNN," ICTACT J. Soft Comput., vol. 13, no. 1, pp. 2805–2812, Oct. 2022, doi: 10.21917/ijsc.2022.0389.

[5] A. Gupta, P. Bhattacharya, S. Agarwal, and P. Goyal, "Agentic AI: Autonomous Intelligence for Complex Goals – A Comprehensive Survey," arXiv preprint arXiv:2504.08148, 2025.

[6] H. Kwon et al., "Automated Business Decision-Making Using Generative AI in Online A/B Testing: Comparative Analysis With Human Decision-Making," in IEEE Access, vol. 13, pp. 124530-124542, 2025, doi: 10.1109/ACCESS.2025.3588480.

[7] C. Pang, "Toward Data Systems That Are Business Semantic Centric and AI Agents Assisted," IEEE Access, vol. 13, pp. 113752–113762, 2025. doi: 10.1109/ACCESS.2025.3583260.

[8] M. H. Qasem, N. Obeid, A. Hudaib, M. A. Almaiah, A. Al-Zahrani, and A. Al-Khasawneh, "Multi-Agent System Combined With Distributed Data Mining for Mutual Collaboration Classification," IEEE Access, 2025.

[9] V. D. Sera and S. Kalra, "Dynamic Architectures Leveraging AI Agents and Human-in-the-Loop for Data Center Management," arXiv preprint arXiv:2504.05427, 2025.

[10] R. Garimella, "Building Multimodal Knowledge Graphs: Automation for Enterprise Integration," arXiv preprint arXiv:2507, 2025.Obstruction avoidance through object detection and classification. IEEE Access, 10:13428–13441, 2022.