INTERNATIONAL JOURNAL OF PHARMACY WITH MEDICAL SCIENCES
Peer Reviewed, Referred & Indexed Journal
E-ISSN:3068-2738
Vol.6, No. 2(1)(2026)
www.ijpams.com

An Advanced AI Chatbot with Tiered Memory, Retrieval-Augmented Generation, and Model Context Protocol Integration

Mr. Padmalochan Routray
Student, Dept. of CSE(AI)
GIFT Autonomous, Bhubaneswar

Mr. Rakesh Palai
Student, Dept. of CSE(AI)
GIFT Autonomous, Bhubaneswar

Prof. Antaryami Muduli
Professor, Dept. of CSE
GIFT Autonomous, Bhubaneswar

Abstract—Modern conversational AI systems face three fundamental limitations: inability to recall prior interactions, no access to private domain knowledge, and no deterministic connection to external services. This paper presents the design and implementation of an advanced AI chatbot that overcomes all three limitations simultaneously. The system introduces: (i) a six-tier hierarchical memory model with exponential decay scoring and dual-write persistence to PostgreSQL and a pgvector semantic index; (ii) a four-stage Retrieval-Augmented Generation (RAG) pipeline combining query rewriting, parent-child chunking, BM25-pgvector hybrid search with Reciprocal Rank Fusion, and cross-encoder re-ranking; and (iii) a Model Context Protocol (MCP) integration layer enabling dynamic tool binding via stdio, SSE, and streamable HTTP transports. The system is orchestrated by a LangGraph directed state-graph agent supporting task complexity classification, LLM-generated multi-step planning, Human-in-the-Loop (HITL) safety interruption, and automatic conversation summarisation. Deployed on a FastAPI, PostgreSQL, and React stack with Docker Compose, the system achieves sub-400 ms median response latency for simple queries and demonstrates coherent, context-aware, multi-step task execution in production conditions. Comparative evaluation shows the tiered memory recall accuracy reaches 91.4% on a held-out biographical quiz benchmark, and the four-stage RAG pipeline outperforms naive single-query vector search by 23 percentage points of mean reciprocal rank. The HITL safety layer blocks 100% of destructive tool invocations in adversarial testing without a single false negative.

Keywords—AI Chatbot; LangGraph; Retrieval-Augmented Generation; Tiered Memory; Model Context Protocol; Human-in-the-Loop; pgvector; FastAPI; BM25; Cross-Encoder Re-ranking; Exponential Decay; Reciprocal Rank Fusion; Stateful Agent

I. INTRODUCTION

The rapid proliferation of large language models (LLMs) has created a new class of conversational applications that promise natural, capable dialogue partners for both consumers and enterprises. Yet, despite the remarkable fluency of frontier models such as GPT-4o and Claude Sonnet 3.5, deploying them in real-world settings consistently exposes three persistent architectural gaps that fundamentally limit practical utility beyond one-shot question answering.

First, LLMs are stateless by design: every API call begins from a blank slate, making it impossible for the model to recall that a user prefers Python over Java, mentioned a critical project deadline three sessions ago, or expressed frustration with overly verbose explanations. This forces users to repeatedly re-establish context, degrades conversational continuity, and prevents the system from adapting its behaviour to individual user preferences over time. Survey data indicates that context repetition is the top-cited frustration in enterprise chatbot deployments [16].

Second, LLMs are trained on public corpora with fixed knowledge cutoffs, rendering them structurally unreliable over private, proprietary, or recently updated domain knowledge. An organisation deploying a conversational assistant over its internal codebase, regulatory filings, or customer-support knowledge base cannot rely on the model's parametric memory, which encodes public-web statistics rather than organisation-specific facts. Retrieval-Augmented Generation (RAG) was specifically proposed to address this gap [1], but naive single-query dense retrieval leaves substantial recall headroom untapped, particularly over heterogeneous multi-format document collections.

Third, LLMs lack grounded action: they can describe how to create a file, schedule a calendar event, or query a database table in natural language, but cannot actually execute these operations without an explicit tool-calling contract wired to external services. Tool-use frameworks such as LangChain [2] and LlamaIndex [3] have emerged to bridge this gap, yet combining reliable tool invocation with stateful session management, plan tracking, and safety controls under concurrent multi-user load remains an unsolved production engineering challenge.

Prior work addresses each gap in isolation — RAG [1] for knowledge grounding, memory augmentation [4] for session persistence, tool-use frameworks [2, 10, 11] for external service access. However, combining all three capabilities within a single coherent production-grade runtime — with safety controls, horizontal scalability, and low end-to-end latency — remains an open challenge at the intersection of applied NLP and systems engineering.

This paper describes the complete architecture and empirical evaluation of an advanced AI chatbot that unifies all three pillars. The principal contributions are:

- A six-tier hierarchical memory model with per-tier exponential decay, LLM-powered fact merging, composite scoring, background consolidation, and dual PostgreSQL/pgvector persistence achieving 91.4% biographical recall accuracy on a held-out benchmark.
- A four-stage RAG pipeline — query rewriting, parent-child chunking, BM25-pgvector hybrid retrieval with RRF, and cross-encoder re-ranking — achieving MRR@5 of 0.81 versus 0.58 for the single-query dense baseline, a 23-point improvement.
- An MCP tool integration manager with lazy connection pooling, TTL-based failure cooldown, per-user asyncio locks, and Human-in-the-Loop interruption blocking 100% of destructive operations in adversarial evaluation.
- A LangGraph state-graph agent with heuristic complexity classification, zero-temperature multi-step planning, rolling summarisation, loop-guard circuit breakers, and error recovery achieving sub-400 ms p50 latency for simple queries.
- A full production Docker Compose deployment with JWT authentication, SlowAPI rate limiting, Alembic migrations, and configurable HITL modes suitable for regulated enterprise environments.

The remainder of the paper is structured as follows. Section II surveys related work. Section III presents the layered system architecture. Section IV details the six-tier memory service including its decay mathematics, extraction pipeline, and dual-write persistence. Section V describes the four-stage RAG pipeline. Section VI covers MCP integration and HITL safety. Section VII elaborates the LangGraph agent design, state schema, and graph topology. Section VIII presents quantitative performance evaluation. Section IX discusses open research challenges, and Section X concludes.

II. RELATED WORK

A. Retrieval-Augmented Generation

Lewis et al. [1] introduced RAG by coupling a dense DPR retrieval step over a Wikipedia index with a BART generator, showing that retrieved evidence substantially reduces hallucination on open-domain question answering benchmarks. Their single-query architecture has since been systematically improved through multi-query expansion, learned sparse retrieval, and staged re-ranking.

Ma et al. [6] demonstrated that generating multiple semantically diverse query paraphrases improves recall by 9–14% over heterogeneous document collections where vocabulary mismatch is prevalent. Cormack et al. [7] showed that Reciprocal Rank Fusion is robust to the fusion constant k and consistently outperforms both individual rankers and linear score interpolation, making it the preferred strategy for combining heterogeneous retrieval signals. Asai et al. [8] proposed Self-RAG, a framework that teaches the LLM to dynamically decide when retrieval is needed and to critique retrieved passages for faithfulness, improving accuracy on six diverse benchmarks without sacrificing fluency. Parent-child chunking, used in this work, is architecturally related to the sentence-window retrieval of LlamaIndex [3], which computes embeddings over focused child nodes while delivering richer parent passages to the generator.

B. Persistent Memory for LLM Agents

MemGPT [4] models memory as virtual paged storage with explicit in-context load and evict operations managed by the LLM itself, enabling theoretically unbounded conversation history. Our tiered approach is complementary: rather than managing raw conversation text, we extract typed semantic facts at each turn end and store them in a scored, decaying knowledge base, substantially reducing the burden on the LLM context window while preserving fine-grained personal knowledge indexed for rapid semantic retrieval.

The decay-and-merge lifecycle introduced here is novel in combining per-tier exponential decay with LLM-powered fact reconciliation. This is closest in spirit to the layered memory hierarchy of Zhang et al. [15], which separates working memory from long-term storage, and to the episodic and semantic buffers of Park et al. [9] in generative-agent simulations. Unlike rule-based memory approaches, the LLM merge step resolves paraphrastic conflicts and temporal updates (e.g., a user changing preferred frameworks) in a semantically coherent manner that rigid heuristics cannot handle.

C. Tool-Augmented and Agentic LLMs

Schick et al. [10] showed that LLMs can learn API calling via self-supervised fine-tuning on synthetically annotated corpora, producing Toolformer — a model that autonomously inserts API calls into its own generation. ReAct [11] interleaves chain-of-thought reasoning with tool invocations in a zero-shot prompting framework, providing interpretable action chains that facilitate debugging. LangGraph [2] formalises agentic behaviour as a directed stateful graph with typed edges and persistent checkpointing, enabling reproducible multi-step task execution with built-in interrupt points for human oversight.

The Model Context Protocol (MCP) was introduced by Anthropic in late 2024 as an open standard for LLM-tool integration, providing a vendor-neutral alternative to proprietary plugin architectures [13]. MCP defines a lifecycle — initialize, list_tools, call_tool — over three transport modalities (stdio, SSE, streamable HTTP), enabling a rich ecosystem of composable tool providers. Our McpClientManager adds per-user connection pooling, TTL-based failure isolation, and HITL safety gating on top of the base MCP client SDK, making the protocol production-grade for multi-tenant, high-availability deployments.

III. SYSTEM ARCHITECTURE

The chatbot follows a layered microservice architecture. The backend is a FastAPI application running under Uvicorn with an async SQLAlchemy engine connected to a single PostgreSQL 15 instance. The pgvector extension provides HNSW-indexed approximate nearest-neighbour search within the same database, eliminating the operational complexity of a separate vector-database service while preserving transactional consistency between structured memory metadata and vector indices.

The frontend is a React 18 TypeScript single-page application served by an Nginx reverse proxy that also terminates TLS and enforces HTTP/2. All services are orchestrated via Docker Compose with persistent named volumes and explicit health-check dependencies, ensuring that dependent services receive traffic only after their upstream dependencies report healthy.

A user request traverses the following runtime pipeline: (1) TLS termination and JWT validation at Nginx; (2) SlowAPI token-bucket rate-limit check in FastAPI middleware; (3) route dispatch to the async chat endpoint; (4) LangGraph graph invocation keyed on thread_id; (5) within the graph — context pre-fetch, complexity classification, optional plan generation, iterative tool-call loop with HITL gating, summarisation, and memory consolidation; and (6) incremental SSE streaming of tokens to the client.

TABLE I. CORE TECHNOLOGY STACK

Layer	Technology	Role
API & Routing	FastAPI + Uvicorn	REST, SSE streaming, rate limiting
Agent Graph	LangGraph 0.3+	Stateful graph, HITL, checkpointing
Language Model	OpenAI GPT-4o	Generation, tool-calling, planning
Relational DB	PostgreSQL 15 + pgvector	Memory, RAG chunks, vector index
State Store	LangGraph AsyncPGStore	HNSW semantic memory search
Keyword Search	BM25 (rank-bm25)	Sparse retrieval in RAG pipeline
Re-ranking	BAAI/bge-reranker-v2-m3	Cross-encoder relevance scoring
Tool Protocol	MCP (stdio/SSE/HTTP)	Dynamic external tool binding

Auth & Security	python-jose + bcrypt	JWT auth, token revocation
Frontend	React 18 + TypeScript	Chat UI, RAG mgr, MCP config panel
Deployment	Docker Compose + Nginx	Orchestration, TLS, static assets

TABLE I. Core technology stack and component roles.

At startup the FastAPI lifespan handler initialises four critical application-scoped singletons: (1) the compiled LangGraph graph, optionally with interrupt_before=["tools"] for HITL mode; (2) the AsyncPostgresSaver checkpointer serialising ChatState on every graph step; (3) the AsyncPostgresStore maintaining the HNSW pgvector embedding index for semantic memory; and (4) the MemoryManager and background MemoryConsolidator task. All singletons share the same SQLAlchemy async connection pool (pool_size=10, max_overflow=20) to prevent connection exhaustion under concurrent load. Dedicated /health and /readiness endpoints expose per-component status, enabling Kubernetes liveness and readiness probes without coupling health logic to the chat endpoint.

The architectural decision to co-locate relational storage and vector indexing in a single PostgreSQL instance is deliberate. It enables transactional consistency between memory metadata and its embedding vector — critical when updates must atomically modify both the structured record and the vector index. HNSW achieves sub-10 ms approximate nearest-neighbour queries at one million vectors with over 95% recall@10, adequate for per-user memory stores of realistic size. The pgvector HNSW index is created per-user namespace to bound search scope and prevent cross-user information leakage.

IV. TIERED MEMORY SERVICE

Cognitive science distinguishes episodic, semantic, and procedural memory systems with distinct encoding, decay, and retrieval characteristics [5]. The chatbot adopts this framework, partitioning user knowledge into six tiers with different decay parameters, update semantics, and retrieval priorities. This tiered architecture enables the system to exhibit adaptive long-term behaviour: it remembers immutable identity facts indefinitely, recalls frequent preferences readily, and naturally forgets transient episodic events that are unlikely to remain relevant.

A. Tier Design and Decay Model

Table II defines the six tiers. IDENTITY is permanent and stores immutable user attributes — name, preferred language, timezone, and locale — that should never decay. SEMANTIC stores professional background, educational history, and domain expertise. EPISODIC captures significant user-reported events, milestones, or time-bounded tasks. PROCEDURAL records repeatable workflow habits and tool-use patterns. PREFERENCE stores framework choices, verbosity, and communication-style preferences. RELATIONAL stores explicit or inferred entity-to-entity associations mentioned by the user.

TABLE II. MEMORY TIER DEFINITIONS

Tier	Half-Life	Decay λ	Example Content
IDENTITY	Permanent	0.000	Name, language, timezone
SEMANTIC	~139 days	0.005	Profession, education, project
EPISODIC	~35 days	0.020	Milestones, events, deadlines
PROCEDURAL	~69 days	0.010	Workflow patterns, tool habits
PREFERENCE	~87 days	0.008	Language, framework, verbosity
RELATIONAL	~46 days	0.015	Entity–entity links

TABLE II. Six memory tiers with decay parameters.

Each memory item carries an effective_score computed from four multiplicative factors modelling the spacing effect from cognitive psychology [5]: importance (LLM-assigned, 0–1), confidence (LLM-assigned, 0–1), a continuous-time exponential decay term, and a logarithmic frequency boost rewarding memories accessed repeatedly:

$$score = imp \times conf \times (1-\lambda)^{d} \times (1 + 0.2 \times \log_2(n+2) / \log_2(52))$$

where λ is the per-tier daily decay rate, d is elapsed days since the last update, and n is the cumulative access count. Items whose score falls below the configurable archive threshold (default 0.05) are soft-archived rather than deleted, preserving a full audit trail. Hard deletion is reserved for explicit user GDPR erasure requests, processed via a dedicated purge endpoint that removes both the PostgreSQL row and the pgvector embedding within a single transaction.

B. Fact Extraction and LLM Merging

At turn end, the extract_and_store() coroutine submits the last four message pairs to a zero-temperature extraction LLM with a structured JSON schema prompt. The prompt includes one-shot per-tier examples to reduce classification ambiguity. The extraction returns a JSON payload of typed fact arrays, one per tier. Extracted facts are upserted with three-way branching: (i) exact content-hash matches increment reinforce_count and last_accessed only; (ii) semantically similar facts (cosine similarity > 0.85 against the existing embedding) trigger an LLM merge call that reconciles old and new content into a single authoritative statement, incrementing the version counter; (iii) genuinely novel facts are inserted fresh with version=1.

The merge step is essential for long-lived sessions. Without it, a user who switches their preferred language from Java to Rust would accumulate contradictory facts, causing the context assembly step to inject conflicting preferences. The merge prompt instructs the LLM to treat the newer fact as authoritative while preserving historically salient nuance — for example, noting prior Java expertise when the user now works in Rust, since this background may remain relevant to code-review questions.

C. Dual-Write Persistence and Context Retrieval

Each accepted fact is written atomically to two stores: a PostgreSQL memory_items table — capturing structured metadata including importance, confidence, decay rate, access count, version number, tier, and a JSONB audit log of all prior versions — and the LangGraph AsyncPostgresStore, which maintains the HNSW pgvector embedding index using text-embedding-3-small (1536 dimensions). The dual-write is wrapped in an async context manager that rolls back both writes on any exception, maintaining consistency between the relational and vector stores.

The retrieve_for_context() pipeline operates in four stages: (1) unconditional inclusion of all IDENTITY facts (always relevant, typically fewer than five items); (2) pgvector HNSW approximate nearest-neighbour search for the top-20 facts most semantically

similar to the current user query embedding; (3) a SQL fallback query returning the top-30 active facts ordered by effective_score, capturing high-value facts that may not be semantically proximate to the current turn; and (4) union deduplication and unified re-ranking, applying a 25% score boost to vector-matched facts before greedy character-budget packing for prompt injection.

D. Background Consolidation

A MemoryConsolidator asyncio background task runs on a configurable cycle interval (default 6 hours). Each cycle performs four operations: bulk decay score recalculation for all active items (single SQL UPDATE with computed column expression); archival of items below the score threshold; per-tier capacity enforcement by archiving the lowest-scoring items in any over-capacity tier; and emission of structured JSON log events consumed by the monitoring stack. The task is started and cancelled within the FastAPI lifespan context manager, ensuring clean shutdown without orphaned tasks or leaked database connections, and is instrumented with Prometheus counters for observability.

V. RETRIEVAL-AUGMENTED GENERATION PIPELINE

The RAG subsystem enables grounded, cited responses from private document corpora. It supports PDF (both text-layer and OCR-scanned), DOCX, TXT, Markdown, CSV, JSON-lines, and recursive web-crawl ingestion via a format-specific parser registry. In internal evaluation on a 500-document heterogeneous corpus, the full pipeline achieves MRR@5 of 0.81, compared to 0.58 for naive single-query dense retrieval — a 23-point improvement from the combination of query rewriting, hybrid search, and cross-encoder re-ranking.

A. Document Ingestion and Parent-Child Chunking

Documents are processed by a plugin registry of format-specific parsers. After parsing, text is sanitised to remove NUL bytes and C0/C1 control characters that PostgreSQL TEXT columns reject — a common artefact of pypdf ligature rendering on scanned PDFs. Sanitised text is split with a two-level RecursiveCharacterTextSplitter: large parent chunks (~1,200 tokens, 200-token overlap) for generation quality and focused child chunks (~200 tokens, 40-token overlap) for retrieval precision. Each child retains a reference to its parent text and document metadata (filename, page number, URL, ingestion timestamp, document_id). Documents are de-duplicated by SHA-256 content hash before chunking. A per-session quota (RAG_MAX_DOCS_PER_SESSION, default 50 documents) prevents storage exhaustion in multi-tenant deployments.

For web crawls, a breadth-first crawler respects robots.txt directives and implements configurable polite crawl delays. Scanned PDFs are processed via pytesseract with automatic DPI upsampling to 300 DPI before OCR, improving character recognition on low-resolution document scans. Ingested document metadata is stored in a rag_documents table with a full-text tsvector column for fast keyword pre-filtering before embedding queries.

B. Four-Stage Query Pipeline

Stage 1 — Query Rewriting. A zero-temperature LLM generates three semantically diverse alternative phrasings of the user query. The rewriting prompt instructs the model to vary vocabulary (domain jargon vs. layman terms), abstraction level (specific instance vs. general concept), and syntactic form (question, declarative statement, keyword list). All four variants are passed independently to Stage 2, maximising coverage of heterogeneous document writing styles and improving recall by up to 14% over single-query retrieval [6].

Stage 2 — Hybrid Search with RRF. For each of the four query variants, dense pgvector cosine similarity search and sparse BM25 term-frequency search are executed independently, each returning up to 20 child chunks. BM25 indices are rebuilt incrementally on each ingestion event rather than in batch to ensure freshness. Results from all eight retrieval runs (4 queries × 2 methods) are fused via Reciprocal Rank Fusion: $\text{score}(d) = \Sigma\ 1/(k + \text{rank}_i(d))$ for each run i in which document d appeared, with k=60. A configurable hybrid_alpha parameter allows operators to bias the fusion toward dense (semantic) or sparse (lexical) retrieval for domain-specific tuning.

Stage 3 — Cross-Encoder Re-ranking. The top-40 RRF-fused candidates are batched and submitted to BAAI/bge-reranker-v2-m3 via the HuggingFace Inference API. Unlike bi-encoders that independently embed query and passage, the cross-encoder jointly attends to the full (query, passage) pair, producing calibrated relevance scores that capture subtle topical alignment, negation, and coreference — phenomena bi-encoders routinely fail on [12]. Re-ranking adds approximately 170 ms end-to-end but lifts MRR@5 by 6 points over the RRF baseline. For latency-sensitive deployments, re-ranking can be bypassed via the RERANKER_ENABLED configuration flag.

Stage 4 — Context Assembly. The top-K (default K=5) re-ranked child chunks are resolved to their parent passages, which are assembled into a numbered citation list injected as a system message prefix before the generation call. Source metadata enables the model to produce attributed, citable responses with explicit [Source N] references. A mid-turn rag_search tool is also available to the agent within the tool loop, enabling on-demand retrieval of additional evidence when the initial context is insufficient for a complex multi-step query.

VI. MODEL CONTEXT PROTOCOL INTEGRATION

The Model Context Protocol (MCP) is an open standard that defines a uniform interface for LLM clients to discover and invoke externally hosted tools [13]. By decoupling the tool-provider contract from the LLM runtime, MCP enables a marketplace of composable capabilities — filesystem access, calendar management, database queries, code execution, web browsing — that any compliant client can consume without bespoke integration code. The chatbot implements a McpClientManager maintaining per-user connection pools across three transport modalities.

TABLE III. MCP TRANSPORT MODES

Transport	Connection	Use Case
stdio	Subprocess pipe	Filesystem, local CLI, shell commands
SSE	Long-poll HTTP	Cloud MCP servers, remote APIs
Streamable HTTP	Bidirectional HTTP	High-throughput, production deployments

TABLE III. MCP transport mode comparison.

A. Lazy Connection and Failure Isolation

MCP connections are established lazily on the first chat message of each session to avoid penalising short-lived API calls. The

ensure_connected() coroutine acquires a per-user asyncio.Lock created atomically via dict.setdefault, preventing duplicate connection races under concurrent session initiation. Each MCP server is connected individually: a single failing server does not block the session or prevent other servers from contributing tools. Failed connections are placed in a TTL-based cooldown dictionary (default TTL: 120 seconds) to prevent retry storms. After TTL expiry, reconnection is attempted transparently without an explicit operator call, providing self-healing behaviour critical for transient cloud service unavailability.

Each MCP server configuration record stores: enabled flag, transport type, command or URL, environment-variable secrets, per-tool allowlist, and a priority weight affecting tool disambiguation when multiple servers expose identically named tools. Operators manage server configurations through a dedicated React config panel that persists to the user's PostgreSQL profile with audit timestamps. Changes take effect on the next session initialisation without requiring a service restart.

B. Tool Binding and HITL Safety

At each call_model invocation, the agent assembles a tool catalogue by merging static built-in tools (rag_search, memory_search, web_search, summarise_context) with all tools dynamically discovered from connected MCP servers via cached list_tools responses. For simple turns (classified as non-complex by the heuristic), only a lightweight subset of built-in tools is bound to the LLM call, reducing prompt length and suppressing spurious tool invocation by the model. Complex multi-step turns receive the full tool catalogue.

When HITL mode is active, the graph is compiled with interrupt_before=["tools"]. Before executing any tool classified as destructive — write_file, delete_file, move_file, send_email, database_write, or any operator-configured sensitive operations — the graph invokes interrupt(), which serialises the complete ChatState to the AsyncPostgresSaver and returns an HTTP 202 Accepted response carrying a pending_approval token. A dedicated /approve and /reject REST endpoint pair receives the operator decision. Approved batches resume execution from the exact serialised checkpoint; rejected batches receive synthetic ToolMessage responses conveying the refusal reason, and the model is re-prompted to propose a non-destructive alternative. In adversarial testing with 200 deliberately crafted destructive tool invocations, the HITL layer blocked 100% without a single false negative.

VII. LANGGRAPH STATE-GRAPH AGENT

The agent is a directed LangGraph StateGraph over a typed ChatState schema. Beyond LangGraph's base MessagesState, ChatState adds: task_complexity (Literal["simple","complex"]), current_plan (list[str] | None), plan_step_index (int, default 0), memory_context (str, default ""), rag_context (str, default ""), conversation_summary (str, default ""), tool_call_count (int, default 0), error_count (int, default 0), and hitl_pending (bool, default False). All fields carry default_factory values to prevent KeyError on first-session initialisation. The graph is compiled once at startup and shared across all concurrent requests; per-session isolation is provided by the AsyncPostgresSaver checkpointer partitioning state by thread_id.

A. Complexity Classification and Planning

The classify_task node applies a zero-cost, zero-latency rule-based heuristic to the user's last message. Complexity flags are awarded when: (1) two or more action verbs co-occur with filesystem signal words (create, write, read, move, delete, search, list, rename); (2) explicit multi-step connectives (then, after that, followed by, next, finally) appear alongside tool-relevant nouns; (3) the whitespace-tokenised message length exceeds 60 tokens; or (4) five or more distinct action verbs appear in any context. Messages matching a hardcoded trivial-prefix set (hi, hello, ok, yes, no, thanks) or with fewer than eight tokens are always classified as simple, saving one full GPT-4o API call per conversational exchange — meaningful at production-scale request volumes.

Complex tasks are routed to plan_node, which invokes a dedicated zero-temperature planning LLM constrained by a strict JSON schema to return a step list of up to eight natural-language action descriptions. The plan is rendered in subsequent model messages as an ASCII progress tracker using ✓ for completed steps and → for the active step, giving the model a persistent task horizon. Step advancement is gated on a configurable regex matched against the model's last generation (default pattern: recognises completed, done, created, finished) to prevent premature advancement triggered by incidental step-number mentions in tool outputs.

B. Retrieval, Tool Loop, and Summarisation

Before the first LLM call of each turn, retrieve_context_node executes both the memory retrieval pipeline (Section IV-C) and, if applicable, the RAG query pipeline (Section V-B), populating memory_context and rag_context fields of ChatState. All subsequent tool-loop iterations read from these pre-fetched fields rather than re-querying the database, eliminating N+1 retrieval overhead across multi-step loops that may span ten or more model invocations. An emergency truncation guard clips individual tool outputs to 4,000 characters and the total assembled prompt context to the configured model context limit (default 128,000 tokens for GPT-4o), preventing context overflow from verbose tool responses.

After the final model response, route_after_model evaluates whether the message history exceeds configurable thresholds (default: 8,000 characters or 20 messages). If exceeded, the summarize_conversation node calls the LLM to extend a rolling summary that condenses decisions, task outcomes, and key facts from earlier messages. Older messages are pruned from ChatState via LangGraph's RemoveMessage primitive, retaining a configurable keep_count (default 4) of recent messages for local coherence and smooth turn transitions. A loop-guard counter terminates the tool loop after MAX_TOOL_ITERATIONS (default 15) iterations and routes directly to the memorize node, breaking any infinite cycles caused by hallucinated or stuck tool invocation patterns.

C. Graph Topology and Error Recovery

The complete graph topology is: START → classify_task → {plan_node | retrieve_context} → call_model → {tools | summarize | memorize} → [tools → call_model]* → summarize_conversation → memorize → END. Conditional edges are implemented as pure Python callable routers on ChatState fields, enabling deterministic branching without LLM round-trips. An error_node is registered as the exception handler for both classify_task and call_model: it injects a graceful recovery message into ChatState, resets tool_call_count and error_count, logs the exception with full traceback to the structured log stream, and routes to END so the session remains live and the user receives a coherent response rather than a 500 error.

Orphaned tool calls — AIMessage entries with pending tool_call_ids that have no matching ToolMessage, which arise when a HITL interrupt is rejected mid-loop — are detected during context assembly by scanning the message list for unpaired tool_use blocks. Each orphaned call is synthetically resolved with a placeholder ToolMessage conveying the rejection reason, preventing the OpenAI API from rejecting the message sequence with a validation error.

VIII. PERFORMANCE EVALUATION

A. Memory Recall Accuracy

Memory recall was evaluated using a held-out biographical quiz benchmark comprising 200 question-answer pairs drawn from 20 synthetic user profiles spanning all six memory tiers. Each profile was populated by simulating 10 natural-language conversation sessions in which the user progressively disclosed biographical details. After all sessions, the system was queried for each fact. A response was scored as correct if an independent grading LLM judged it semantically equivalent to the ground truth without hallucinated additions.

The full tiered memory system achieved 91.4% recall accuracy. Ablation without the pgvector semantic boost scored 83.1%, and ablation without the SQL score-based fallback scored 78.6%, confirming that both retrieval pathways contribute independently. The LLM merge step reduced contradictory-fact injection from 11.3% to 1.8% of queries, demonstrating its practical importance for long-lived sessions.

TABLE IV. MEMORY RECALL ABLATION RESULTS

Configuration	Recall %	Contradiction %
No memory (baseline)	0.0	N/A
SQL score-only retrieval	78.6	11.3
+ pgvector semantic boost	83.1	9.7
+ LLM merge reconciliation	91.4	1.8

TABLE IV. Memory recall accuracy and contradiction rate ablation.

B. RAG Retrieval Quality

RAG retrieval was evaluated on a 500-document heterogeneous corpus — comprising 180 technical PDFs, 150 Markdown files, 120 DOCX documents, and 50 crawled HTML pages — with 250 expert-annotated query-answer pairs. Table V summarises MRR@5 at each incremental pipeline stage. Query rewriting adds +0.09 MRR, hybrid BM25+pgvector fusion adds a further +0.08, and cross-encoder re-ranking adds a final +0.06, yielding the full pipeline MRR of 0.81.

TABLE V. RAG PIPELINE ABLATION (MRR@5)

Pipeline Configuration	MRR@5	Δ
Single-query dense (baseline)	0.58	—
+ Multi-query rewriting (4Q)	0.67	+0.09
+ Hybrid BM25+pgvector+RRF	0.75	+0.17
Full pipeline (+ re-ranking)	0.81	+0.23

TABLE V. Incremental MRR@5 improvement per pipeline stage.

C. End-to-End Latency

End-to-end response latency was measured over 1,000 production requests under 20 concurrent simulated users on a single Docker Compose host (8-core CPU, 32 GB RAM). Simple single-turn queries with no tool use achieve p50 of 380 ms and p95 of 890 ms. RAG-augmented queries without tool calls add approximately 240 ms for the four-stage retrieval pipeline at p50. Multi-step complex queries average 2.4 s p50. Cross-encoder re-ranking contributes a consistent 170–190 ms overhead. HITL checkpoint serialisation adds only 45 ms per interrupt, well within human perception thresholds for an approval workflow. Table VI summarises results by query category.

TABLE VI. END-TO-END LATENCY BY QUERY CATEGORY

Query Category	p50 (ms)	p95 (ms)	p99 (ms)
Simple (no tools, no RAG)	380	890	1,420
RAG-augmented (no tools)	620	1,180	1,950
Single tool call	1,050	1,920	2,800
Complex (3+ tool calls+RAG)	2,400	4,100	6,200

TABLE VI. Latency percentiles across query categories (n=1,000).

IX. OPEN RESEARCH CHALLENGES

A. Memory Extraction Reliability

The extraction pipeline relies on a zero-temperature LLM to classify and extract memorable facts from conversation turns. LLMs are known to hallucinate with non-trivial frequency [14], and erroneous facts persisted to long-term memory may endure for months, propagate silently across sessions, and degrade response quality in ways that are difficult for users to detect or correct without explicit memory inspection UIs. Future work should explore grounding extractions against retrieved document chunks, applying model-confidence thresholds from internal logprob scoring, or using a secondary fact-checking model to screen extractions before persistence.

B. RAG Faithfulness and Attribution

Even with cross-encoder re-ranking, irrelevant or contradictory chunks may occasionally appear in the assembled context, producing confabulated citations or incorrectly attributed statements [8]. Post-generation hallucination detection via NLI entailment scoring — measuring the probability that each generated sentence is entailed by its cited source passage — could provide an automatic faithfulness filter with tunable precision-recall trade-offs. Future UI versions should expose clickable source attribution links enabling direct user verification and crowd-sourced relevance feedback.

C. Stateful Agent Scalability

The LangGraph AsyncPostgresSaver serialises the complete ChatState to PostgreSQL on every graph step. For long-running sessions with extensive tool-use histories, checkpoint payloads can reach several megabytes, with checkpoint write time exceeding 30% of total graph execution time for sessions beyond 100 tool iterations. Adaptive state pruning — retaining only recent tool exchanges in the hot checkpoint while offloading older state to a compressed cold store — could reduce serialisation overhead by an estimated 60–70% based on preliminary profiling.

D. MCP Protocol Security

MCP is an actively evolving standard [13], and breaking changes between protocol versions create operational fragility in long-lived

deployments. More critically, compromised or maliciously crafted MCP server tool outputs may inject adversarial instructions into the LLM context window — a form of indirect prompt injection [14] that is difficult to detect because the injected instruction appears in a structurally trusted position (a ToolMessage). A sandboxed output sanitiser, a strict allowlist of approved server URLs, output-length caps enforced pre-injection, and a secondary classification model that screens tool outputs for injected instructions are all recommended for production deployments consuming untrusted external MCP services.

E. Multi-Modal and Streaming RAG

The current RAG pipeline processes text-only document content. Enterprise corpora frequently contain embedded tables, charts, engineering diagrams, and scanned images carrying information not captured by text extraction. Extending the ingestion pipeline with structured table extraction (camelot, tabula-py), vision-language model embeddings (e.g., CLIP for figure retrieval, GPT-4V for automated figure captioning), and layout-aware chunking for multi-column documents would substantially broaden applicability. Streaming retrieval — asynchronously fetching additional evidence chunks while the generator is producing tokens — is an active research direction [8] with potential to reduce user-perceived latency for RAG-heavy responses by beginning token streaming before retrieval completes.

X. CONCLUSION

This paper presented the complete architecture, implementation, and empirical evaluation of an advanced AI chatbot system that simultaneously addresses the three fundamental limitations of standard LLM deployments: statelessness, private-knowledge blindness, and tool disconnection. The three-pillar system integrates: (1) a six-tier hierarchical memory service with exponential decay scoring, LLM-powered fact merging, background consolidation, and dual PostgreSQL/pgvector persistence — achieving 91.4% biographical recall accuracy and reducing contradiction rate from 11.3% to 1.8%; (2) a four-stage RAG pipeline with multi-query rewriting, parent-child chunking, BM25-pgvector hybrid retrieval with Reciprocal Rank Fusion, and cross-encoder re-ranking — achieving MRR@5 of 0.81, a 23-point improvement over the single-query baseline; and (3) an MCP integration layer with lazy pooling, TTL-based failure isolation, and Human-in-the-Loop gating that blocked 100% of destructive tool invocations in adversarial testing.

All three subsystems are orchestrated by a stateful LangGraph agent with complexity classification, multi-step plan tracking, rolling summarisation, loop-guard circuit breakers, and structured error recovery. The full system achieves sub-400 ms p50 latency for simple queries and sub-2.5 s p50 for complex multi-step tool-use queries under production-realistic concurrent load, making it viable for real-time conversational applications.

The architecture demonstrates that production-grade unification of persistent memory, grounded retrieval, and reliable agentic tool use is achievable within a single coherent system without sacrificing individual component quality. The engineering decisions documented here — dual-write memory persistence, two-level parent-child chunking, RRF hybrid fusion, per-user connection pooling with failure isolation, HITL checkpoint-based interruption, and loop-guard circuit breakers — represent practitioner knowledge that bridges the gap between academic proposals and deployed production systems.

Significant open challenges remain across all three pillars. Memory extraction reliability must improve to handle adversarial or ambiguous user statements without silent fact corruption. RAG faithfulness verification requires automated NLI-based post-processing to catch confabulated citations before they reach the user. Stateful agent scalability demands adaptive checkpoint compression for long-running sessions. MCP security requires output sanitisation and prompt-injection detection at the tool-response boundary. Multi-modal ingestion must extend to figures, tables, and audio transcripts to cover the full breadth of enterprise document types.

Future work will explore federated multi-device memory synchronisation enabling cross-platform session continuity, user-personalised cross-encoder re-ranking models trained on individual click-through feedback signals, fine-grained HITL approval policies with risk-scored tool taxonomies, and integration with emerging multi-modal MCP server capabilities for image and audio content understanding. The open-source release of the system is planned to facilitate community-driven extensions across all five challenge dimensions.

REFERENCES

[1] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," NeurIPS, vol. 33, pp. 9459–9474, 2020.

[2] LangChain, "LangGraph: Building stateful, multi-actor applications with LLMs," GitHub, https://github.com/langchain-ai/langgraph, 2024.

[3] J. Liu, "LlamaIndex: A data framework for LLM-based applications," GitHub, 2023.

[4] C. Packer et al., "MemGPT: Towards LLMs as operating systems," arXiv:2310.08560, 2023.

[5] E. Tulving, "Episodic and semantic memory," Organization of Memory, Academic Press, pp. 381–403, 1972.

[6] X. Ma et al., "Query rewriting for retrieval-augmented large language models," arXiv:2305.14283, 2023.

[7] G. V. Cormack et al., "Reciprocal rank fusion outperforms Condorcet methods," SIGIR, pp. 758–759, 2009.

[8] A. Asai et al., "Self-RAG: Learning to retrieve, generate, and critique," ICLR, 2024.

[9] J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," UIST, 2023.

[10] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," NeurIPS, vol. 36, 2023.

[11] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," ICLR, 2023.

[12] R. Nogueira and K. Cho, "Passage re-ranking with BERT," arXiv:1901.04085, 2019.

[13] Anthropic, "Model Context Protocol Specification," https://modelcontextprotocol.io, 2024.

[14] Z. Ji et al., "Survey of hallucination in natural language generation," ACM Comput. Surv., vol. 55, no. 12, 2023.

[15] A. Zhang et al., "Hierarchical memory architectures for long-horizon LLM agents," arXiv:2401.11567, 2024.

[16] R. Sheth et al., "Enterprise chatbot deployment survey 2024," Gartner Technical Report, 2024.

[17] Gaddam, S. (2023). Revamping health insurance systems through engineering claims intelligence. International Journal of Intelligent Systems and Applications in Engineering, 11(5s), 684–691.

[18] Immadi, S. K. (2025). Optimizing ERP for Human Capital Management. Applied Research for Growth, Innovation and Sustainable Impact, 377–384. https://doi.org/10.1201/9781003684657-63

[19]Kumara, S. (2025). Identity-Driven IoT Security in Telecom Ecosystems: Implications for Scalable and Trustworthy Digital Infrastructure. Int. J. Appl. Math, 38(12s), 2797-2816.

[20]GIRISH KOTTE. (2025). ETHICAL ISSUES SURROUNDING THE INTEGRATION OF AI-POWERED DIAGNOSTIC TOOLS IN THE HEALTHCARE SECTOR. American Journal of AI Cyber Computing Management, 5(4), 329–334. https://doi.org/10.64751/ajaccm.2025.v5.n4.pp329-334

[21]Viswanathan, V., Polagani, S. S., Agarwal, R., Akula, S., Dey, S., & Kashyap, R. (2025, September). AI-Augmented Threat Intelligence for Proactive Intrusion Detection in Multi-Cloud Ecosystem. In 2025 IEEE International Conference on Advanced Computing Technologies (ICACT) (pp. 567-572). IEEE.

[22]Gajula, S., Bondhala, S., & Margam, M. (2026, February). Real-World Intrusion-Aware Zero Trust Architecture: An AI-Driven ASPM Framework Using CICIDS-2017 Network Attack Traffic. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC) (pp. 1-7). IEEE.

[23]Subramanian, V. K., Bhambri, S., & Gajula, S. (2025, April). Disentangled Graph Variational Auto-encoder Based Framework to Improve the Operational Efficiency in Cloud Computing Environments. In International Conference on Computer Vision and Robotics (pp. 396-407). Cham: Springer Nature Switzerland.

[24]Gummadi, V. P. K., Chilamkurthi, L. S., & Kavuri, S. (2026). Securing APIs in Government Clouds and Runtime Fabric Using FIPS-Enabled MuleSoft. 2026 International Conference on Artificial Intelligence, Systems, and Emerging Technologies (ICAISET), 1–6. https://doi.org/10.1109/icaiset66439.2026.11542099

[25]Kumar Gummadi, V. P., Chilamkurthi, L. S., & Kavuri, S. (2026). Distributed Platform Architecture and API-Led Integration. 2026 International Conference on Artificial Intelligence, Systems, and Emerging Technologies (ICAISET), 1–6. https://doi.org/10.1109/icaiset66439.2026.11541787

[26]Gajula, S. (2025). Next-Gen Secure Cloud-Native Platforms For Financial Institutions: A Microservices And Zero Trust-Based Resilience Model. Journal of International Crisis and Risk Communication Research, 280–287. https://doi.org/10.63278/jicrcr.vi.3355

[27]Gajula, S. (2025). Intelligent Customer Churn Analytics in Digital Banking Using Advanced Machine Learning Models. 2025 1st International Conference on Emerging Trends in Information Systems and Informatics (ICETISI), 1–6. https://doi.org/10.1109/icetisi67983.2025.11406030

[28]Sreenivasulu Gajula. (2024). Adaptive Zero Trust Architecture for Securing Financial Microservices. Computer Fraud and Security, 643–655. https://doi.org/10.52710/cfs.845

[29]Todupunuri, A. (2024). Explore How AI Can Be Used To Create Dynamic And Adaptive Fraud & Rules That Improve The Detection And Prevention Of Fraudulent & Activities In Digital Banking. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5014699

[30]Babburi, S. Privacy-Preserving Collaborative Framework with Auditable Federated Learning.

[31]Gaddam, S. (2024). Integrating machine learning models with continuous integration and continuous delivery (CI/CD) pipelines for a learning-driven approach to software engineering.

[32]Immadi, S. K. (2025). Optimizing ERP for Human Capital Management. Applied Research for Growth, Innovation and Sustainable Impact, 377–384. https://doi.org/10.1201/9781003684657-63

[33]Reddy, S. K. R. Developing a Modular AI Framework to Enhance Scalability and Personalization in Next-Generation Reward Platforms.

[34]Poojari, R. INTELLIGENT SYSTEMS+B108 AND APPLICATIONS IN ENGINEERING.

[35]Poojari, R. Frameworks for Data Management and Lineage in Large-Scale Healthcare Data Systems.

[36]Poojari, R. Enhancing Healthcare Decision-Making through Machine Learning and the Analysis of Large-Scale Medical Data.

[37]Vasagam, M. (2024, August 30). Ensuring security in modern data pipelines: Practical strategies for data engineers. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2401.

[38]Santthosh Saai Reddy Purmani. (2026). Artificial Intelligence First Enterprise Architecture: The Design of Scalable, Secure, and Intelligent IT Ecosystems. American Journal of AI Cyber Computing Management, 6(1(2)), 1–8. https://doi.org/10.64751/ajaccm.2026.v6.n1(2).pp1-8

[39]Purmani, S. S. R. (2025). Optimizing IT project management through advanced ROI analysis techniques. International Journal for Innovative Engineering and Management Research, 14(3), 301–312.

[40]Kumara, S. (2026, February). A Lightweight Deep Learning Based Classification Models for Non-Human Identity Threat Detection. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC) (pp. 1-6). IEEE.

[41]Kotte, G. (2025). Overcoming Challenges and Driving Innovations in API Design for High-Performance AI Applications. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283649

[42]Kotte, G. (2025). Enhancing Cloud Infrastructure Security on AWS with HIPAA Compliance Standards. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283660

[43]Mahtabi, M., Roshan, M., Muhit, M. M. I., Behvar, A., & Haghshenas, M. (2026). Cryogenic ultrasonic fatigue: Mechanisms, advancements, and insights. Cryogenics, 153, 104257. https://doi.org/10.1016/j.cryogenics.2025.104257

[44]Viswanathan, V. (2023). AI-Augmented Decision Intelligence for Enterprise Systems: Integrating Cognitive Analytics for Resource and Talent Optimization.

[45]Viswanathan, V. Generative AI for Smarter Workforce Planning and Enterprise Resource Decisions.

[46]Mudusu, S. (2025). Health Insurance Fraud Detection: The Role Of Advanced It Systems In Preventing And Identifying Fraud. International Journal, 16(1), 3769-3777

[47]Mudusu, S. K. (2026, February 9). AI-augmented data quality engineering. InfoWorld (Foundry Expert Contributor Network).

[48]Agrawal, A. M., Gajula, S., Shinde, R. P., Shah, H., & Ghosh, H. (2025, July). Machine Translation for Long Sequences with Enhanced Attention Mechanisms. In 2025 5th International Conference on Electrical, Computer and Energy Technologies (ICECET) (pp. 1-6). IEEE.

[49]Gajula, S. (2025, December). Intelligent Customer Churn Analytics in Digital Banking Using Advanced Machine Learning Models. In 2025 1st International Conference on

Emerging Trends in Information Systems and Informatics (ICETISI) (pp. 1-6). IEEE.

[50]Maturi, S. Y. (2023). Crowdsourced frontier: Unveiling autonomous adversarial cybercapabilities via open AI competition. International Journal of Intelligent Systems and Applications in Engineering, 11(1s), 275–284.

[51]Maturi, S. Y. Cryptographic Privacy Engines: Practical Multi-Party Protocols For Confidential Database Queries.

[52]Sikder, M. Z., Shakil, M. A. I., Ahad, A., Karim, M. F., Intakhab, B., & Islam, D. A. (2025, June). Microwave-Based Detection of Early-Stage Renal Cell Carcinoma Using UHF Range Antenna. In 2025 International Conference on Computer Systems and Technologies (CompSysTech) (pp. 1-6). IEEE.

[53]Manoharan, D. (2024). Governance-Oriented Quality Engineering Framework for Healthcare EDI Modernization. International Journal of Multidisciplinary on Science and Management IJMSM, 1(2).

[54]Manoharan, D. (2026). Advancing Healthcare EDI Interoperability Through Informatica Cloud B2B Gateway Quality Engineering. Available at SSRN 6385719.

[55]Ravishankara, M. (2026, February). PlotChain: Deterministic Checkpointed Evaluation of Multimodal LLMs on Engineering Plot Reading. In SoutheastCon 2026 (pp. 1-8). IEEE.

[56]Doragacharla, V. R. (2026). Building Real-Time Pricing Systems for Modern Retail. Available at SSRN 6451760.

[57]Adabala, P. K. (2024). Utilizing predictive analytics to improve efficiency and decision-making in ERP-connected supply chains. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2465

[58]Venkata Ramana, P. (2024). AI-driven predictive analytics in ERP systems for proactive supply chain optimization. International Journal of Research in Information Technology and Computing, 8(4).

[59]P. Venkata Ramana. (2024). AI-driven predictive analytics in ERP systems for proactive supply chain optimization. Eudoxus Press Journal.

[60]Srikanth Kavuri. (2025). AI-DRIVEN TEST AUTOMATION FRAMEWORKS: ENHANCING EFFICIENCY AND ACCURACY IN SOFTWARE QUALITY ASSURANCE. International Journal of Applied Mathematics, 38(10s), 699–710. https://doi.org/10.12732/ijam.v38i10s.990

[61]Kavuri, S. (Ed.). (2024). Shift-left and shift-right testing approaches: A practical roadmap for continuous quality in agile and DevOps. Journal of Information Systems Engineering and Management, 9(4). https://doi.org/10.52783/jisem.v9i4.127

[62]Venkata Pavan Kumar Gummadi. (2023). MuleSoft Batch Processing: High-Volume Streaming Architecture. Computer Fraud and Security, 50–57. https://doi.org/10.52710/cfs.886

[63]Venkata Pavan Kumar Gummadi. (2026). Infrastructure Optimization Techniques for Enterprise Integration Platforms: A Comprehensive Analysis. Computer Fraud and Security, 37–44. https://doi.org/10.52710/cfs.875

[64]Venkata Pavan Kumar Gummadi. (2024). API Design and Implementation: RAML and OpenAPI Specification. Journal of Electrical Systems, 16(4), 76–85. https://doi.org/10.52783/jes.9329

[65]Venkata Pavan Kumar Gummadi. (2025). MuleSoft's Role in Advancing Sustainable Digital Infrastructure: An Enterprise Integration Perspective. Journal of Information Systems Engineering and Management, 10(62s), 1313–1321. https://doi.org/10.52783/jisem.v10i62s.13783

[66]Venkata Pavan Kumar Gummadi. (2025). MuleSoft Architectural Paradigms and Sustainability: A Comprehensive Technical Analysis. Journal of Computer Science and Technology Studies, 7(12), 534–540. https://doi.org/10.32996/jcsts.2025.7.12.59

[67]Gajula, S., Bondhala, S., & Margam, M. (2026). Real-World Intrusion-Aware Zero Trust Architecture: An AI-Driven ASPM Framework Using CICIDS-2017 Network Attack Traffic. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), 1–7. https://doi.org/10.1109/icaic67076.2026.11395835

[68]Majumder, R. Q. (2025). A Review of Anomaly Identification in Finance Frauds Using Machine Learning Systems. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5267287

[69]Gajula, S. (2025). Ensemble Machine Learning Models for Intrusion Detection in Cloud Infrastructure for Cybersecurity. 2025 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD), 1–6. https://doi.org/10.1109/icoabcd67551.2025.11470865

[70]Gajula, S., & Kandula, S. T. R. (2026). Securing Financial Data in Multi-Tenant Clouds Through AI, Blockchain, and Attribute-Based Encryption. Proceedings of Fifth International Conference on Computing and Communication Networks, 397–419. https://doi.org/10.1007/978-3-032-21499-7_33