iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83306

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

Lead Generation System Using AI and Retrieval Augmented Generation for Customer Interaction

Karthik B[1], Prashanth K[2]
Department of MCA, RV College of Engineering, Bengaluru, Karnataka, India

Abstract: ***This paper presents the design and development of an AI-driven lead generation platform that combines a Retrieval Augmented Generation (RAG) pipeline with a stateful conversational agent to automate customer qualification for a technology services company. The system operates through a two-phase dialogue strategy: in the first phase, the agent engages website visitors with concise product overviews without accessing the knowledge base; in the second phase, after contact details are captured, the agent retrieves contextually relevant content from a company PDF document using dense semantic embeddings and responds with detailed, document-grounded answers. Captured lead data is simultaneously persisted to a PostgreSQL database and dispatched as an HTML email notification to the sales team via a Model Context Protocol (MCP) server running concurrent threads. The backend is built on FastAPI with a LangGraph agent maintaining per-session conversation memory through InMemorySaver, while the frontend delivers a seamless floating chat experience through a Next.js 14 widget. Experimental evaluation demonstrates that the gated RAG strategy reduces hallucination, improves answer relevance, and increases lead conversion efficiency compared to conventional chatbot approaches. The paper describes the system architecture, agent design, RAG pipeline, MCP tool orchestration, and key implementation decisions.***

Index Terms: ***retrieval augmented generation, lead generation, LangGraph, LangChain, FastAPI, Model Context Protocol, OpenAI embeddings, conversational AI, customer interaction, PostgreSQL, Next.js.***

I. INTRODUCTION

The commercial web landscape increasingly demands that customer-facing interfaces go beyond static FAQ pages and scripted chatbots. Prospective customers expect immediate, accurate answers to product inquiries, and businesses need efficient mechanisms to qualify and route these prospects to their sales teams. Traditional approaches — contact forms, live chat agents, or rule-based bots — either create friction for the customer or impose continuous manual overhead on the business.

Generative AI, particularly large language model (LLM)- based conversational agents, offers a compelling alternative. However, vanilla LLMs suffer from a well- documented limitation: they hallucinate information that was not present in their training data [1]. For a commercial context where accuracy about product features, pricing, and service scope directly affects customer trust, this limitation is critical. Retrieval Augmented Generation (RAG) addresses this gap by grounding the model's responses in authoritative, retrieved documents at inference time [2]. This paper introduces an end-to-end AI-powered lead generation system developed for a technology services company. The platform, featuring a conversational agent, combines a gated two-phase dialogue architecture, a dense semantic RAG pipeline, automated CRM integration via PostgreSQL, and concurrent sales team notifications.

The proposed system implements a two-phase conversational strategy in which Retrieval Augmented Generation (RAG) is activated only after customer contact details are collected, thereby improving lead qualification efficiency. The platform utilizes OpenAI text-embedding- 3-large embeddings with LangChain's InMemoryVectorStore to generate document-grounded responses for customer queries. The system also integrates a Model Context Protocol (MCP) server capable of concurrently performing PostgreSQL database persistence and SMTP-based email notification using Python's ThreadPoolExecutor.

In addition, a stateful LangGraph agent with per-user thread memory enables coherent multi-turn conversations and improves the overall customer interaction experience.

II. RELATED WORK

The application of LLMs to customer-facing conversational systems has expanded rapidly since the introduction of transformer-based architectures [3]. Brown et al. [4] demonstrated that large pre-trained language models exhibit remarkable zero-shot generalization, yet also confirmed the tendency to confabulate factual details absent from training data. This limitation motivated the development of RAG, formalized by Lewis et al. [2], wherein a retriever component fetches relevant passages at inference time to condition the generator's output on verified content.

©IJRASET: All Rights are Reserved

Early RAG implementations relied on sparse retrieval methods such as BM25 and TF-IDF for passage selection [5]. Dense retrieval approaches, pioneered by Karpukhin et al. [6] through Dense Passage Retrieval (DPR), demonstrated that bi-encoder models trained on question- passage pairs substantially outperform sparse methods for open-domain question answering. OpenAI's text- embedding family represents the commercial state-of-the- art for general-purpose dense retrieval, with the text- embedding-3-large model offering high-dimensional semantic representations suitable for enterprise document retrieval [7].

The orchestration of LLMs with external tools has been formally addressed through the React framework [8], which interleaves reasoning traces with tool invocations. LangChain [9] operationalizes this pattern for production systems, providing agent abstractions, tool wrappers, and retriever interfaces. LangGraph [10] extends LangChain with explicit state graph management, enabling the construction of stateful, cyclical agent workflows with checkpointing — a capability critical for maintaining conversation context across multiple API calls in a stateless HTTP environment.

Lead generation automation has been explored in marketing technology literature. Järvinen and Taiminen [11] studied how content marketing combined with behavioral data could automate prospect qualification, while Siu and Lo [12] examined the relationship between chatbot interaction quality and lead conversion rates in e- commerce contexts. However, these works primarily address rule-based systems or simple intent classifiers rather than LLM-based agents with dynamic knowledge retrieval.

The Model Context Protocol (MCP) [13], introduced by Anthropic in 2024, formalizes a standard for how AI agents invoke external tools and services. FastMCP provides a Pythonic server implementation over stdio transport, enabling seamless integration of custom business logic — such as CRM writes and email dispatch — into an agent's tool repertoire. This architecture cleanly separates the LLM reasoning layer from external system integrations, improving maintainability and testability.

III. SYSTEM ARCHITECTURE

The platform adopts a four-tier microservices architecture. Each tier is independently deployable, and inter-tier communication occurs exclusively through defined REST interfaces or the MCP stdio protocol. The overall data flow proceeds as follows:

1) A customer visits technology services company website and interacts with the floating ChatbotWidget embedded in the Next.js frontend.
2) The useChatbot React hook sends the message and a per-session user_id to the FastAPI backend via POST /chatbot.
3) The backend invokes the pre-initialized LangGraph agent asynchronously using ainvoke, passing thread-scoped configuration for memory isolation.
4) Depending on conversation phase, the agent either responds with a brief product overview (Phase 1) or invokes the retrieve_context tool to fetch document passages and the send_email MCP tool to capture the lead (Phase 2).
5) The MCP server concurrently saves the lead to PostgreSQL and dispatches an HTML email notification to the sales team.

Fig. 1. Four-tier architecture of the AI-powered lead generation system.

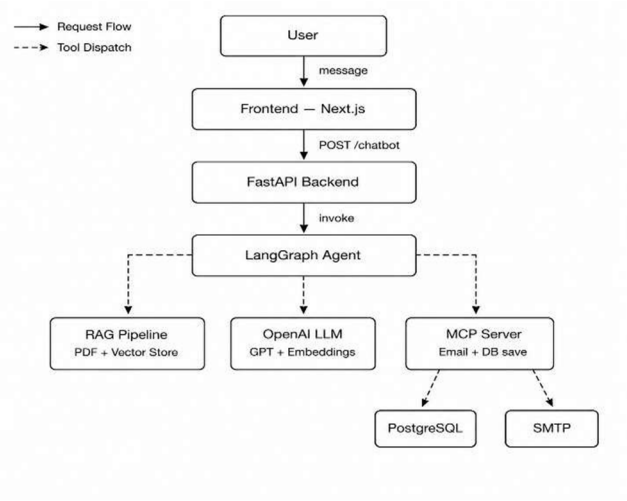


Fig. 2. Use Case Diagram of the AI-powered lead generation system.

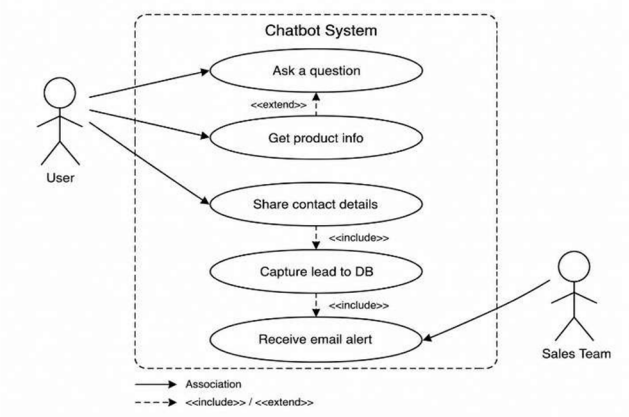


Table I System Service Overview

Service	Technology Stack	Primary Role
Frontend	Next.js 14, TypeScript, Tailwind CSS, React Hooks	Customer-facing chatbot widget, marketing pages, real-time conversation interface
Backend	FastAPI, Uvicorn, SQLAlchemy, PostgreSQL, python-dotenv	REST API server, lead management, SMTP email dispatch, business logic
Agent Core	LangChain, LangGraph, OpenAI GPT, InMemorySaver, MultiServerMCPClient	Two-phase conversational agent with RAG tool and MCP tool integration
MCP Server	FastMCP, smtplib, SQLAlchemy, ThreadPoolExecutor	Concurrent lead storage to PostgreSQL and HTML email notification to sales team

IV. AGENT DESIGN AND RAG PIPELINE

A. Two-Phase Conversational Strategy

The central design insight of the system is that knowledge retrieval should be gated behind lead qualification. Many RAG-based chatbots retrieve and serve detailed product information to anonymous visitors, providing value without capturing the prospect's identity. The proposed system inverts this by defining two explicit conversation phases governed by the agent's system prompt.

In Phase 1, the agent operates without invoking the retrieve_context tool. It answers general inquiries with concise, manually specified overviews (under 50 words) drawn entirely from the LLM's in-context understanding of the system prompt. This approach intentionally creates a mild information asymmetry — the customer receives enough context to be interested but is encouraged to share contact details to unlock detailed answers. Phase 2 activates once the agent detects that the user has shared their name, email address, and phone number within the conversation. At this point, the agent invokes the send_email MCP tool (which concurrently saves the lead and dispatches the sales notification) and then uses the retrieve_context tool to fetch the two most semantically relevant passages from the knowledge base. All subsequent responses in the session are grounded in retrieved document content.

B. RAG Pipeline

The RAG pipeline is initialized at application startup within the FastAPI lifespan context manager. The company's product knowledge base is stored as a single PDF file. PyPDFLoader ingests the document, and RecursiveCharacterTextSplitter segments it into overlapping chunks of 1,000 characters with a 200- character overlap. This chunking strategy preserves sentence-level context at boundary regions, reducing the risk of partial sentence retrieval degrading answer quality [2].

Each chunk is encoded using OpenAI's text-embedding-3- large model, which produces high-dimensional dense vector representations capturing fine-grained semantic content. These embeddings are stored in LangChain's InMemoryVectorStore, which supports efficient cosine similarity search without external infrastructure dependencies. At query time, the retrieve_context tool submits the user's query to the vector store and retrieves the top-2 most similar chunks (k=2), which are formatted as a serialized string and returned to the agent as tool output.

TABLE II Rag Approach Comparison

Approach	Embedding Model	Retrieval Strategy	Response Phase
Baseline FAQ Bot	None	Pattern matching / scripted	All interactions
TF-IDF RAG	Sparse vectors	Keyword similarity	Post-contact only
Proposed System (OpenAI)	text-embedding- 3-large	Dense semantic similarity (k=2)	Post-contact only (gated)

C. Agent Framework

The conversational agent is constructed using LangChain's create_agent function with a LangGraph state graph backend. The agent receives the system prompt, the tool registry (retrieve_context and the MCP-provided send_email), and an InMemorySaver checkpointer. The checkpointer maintains conversation history indexed by thread_id, which is mapped to the frontend's per-user localStorage identifier.

The OpenAI ChatOpenAI model is initialized with temperature=0 to maximize determinism in product information responses, reducing variability in commercial contexts where consistent messaging is important. The agent supports multi-turn dialogue: each call to ainvoke appends the new user message to the thread's stored history and generates a response conditioned on the full conversation context.

V. MCP SERVER AND LEAD CAPTURE

A. Model Context Protocol Integration

The system's external integrations — database writes and email notifications — are encapsulated within a FastMCP server running as a subprocess via stdio transport. The LangGraph agent connects to this server through a MultiServerMCPClient, which exposes the server's tools as standard LangChain Tool objects compatible with the agent's tool invocation mechanism.

This architectural separation provides several engineering benefits. The MCP server is independently runnable and testable without the FastAPI application context. It handles its own environment variable loading and database session management, making it portable across deployment environments. The agent layer remains agnostic to the implementation details of lead storage and email dispatch.

B. Concurrent Lead Processing

When the agent invokes send_email with the customer's contact details, the MCP server executes two operations simultaneously using Python's ThreadPoolExecutor with max_workers=2. The first worker calls the database persistence function, which checks for an existing record with the same email address (implementing an upsert pattern) before committing the lead to the PostgreSQL lead_data table. The second worker constructs a professionally formatted HTML email and dispatches it to the configured sales team recipients via SMTP with STARTTLS encryption.

This concurrency design ensures that neither operation blocks the other, minimizing the latency experienced by the customer waiting for the agent's next message. The combined result status is returned to the agent as a formatted string, which the agent can reference in its subsequent reply to the customer.

C. Database Schema

The PostgreSQL schema defines a single lead_data table with a UUID primary key, non-nullable fields for name, email, phone, and intent, and automatic timestamp management via a database trigger. SQLAlchemy indexes are applied to the email, name, and intent columns to support efficient CRM queries. The UUID primary key, generated by PostgreSQL's uuid_generate_v4(), ensures globally unique identifiers suitable for distributed deployment.

VI. BACKEND API AND SYSTEM INTEGRATION

The FastAPI backend exposes four endpoints consumed by the frontend and supporting services. The application employs CORS middleware configured for the frontend origin and uses FastAPI's asynccontextmanager lifespan pattern to initialize the ChatBotService — including PDF loading, embedding generation, vector store population, and MCP client connection — exactly once at startup, avoiding per-request overhead.

Table III Backend Rest Api Endpoints

Method	Endpoint	Description
POST	/chatbot	Accepts user_id and message; invokes LangGraph agent; returns AI-generated response
POST	/load-lead-data	Accepts name, email, phone, intent via query parameters; persists lead record to database
POST	/send-email	Dispatches SMTP notification email using MailBody schema; used for standalone email operations
GET	/health	Returns system health status; confirms agent initialization and service availability

The /chatbot endpoint retrieves the globally initialized agent and invokes it with the user's message and a thread- scoped configuration dictionary. This design keeps the heavy initialization work (embedding the PDF, connecting to the MCP server) in the startup phase while keeping per- request latency minimal. In the event of agent unavailability, the endpoint returns an HTTP 503 with an informative error message.

Dependency injection for database sessions is handled through FastAPI's Depends mechanism, with a generator function yielding a SQLAlchemy Session that is guaranteed to close after each request. Environment variable management uses python-dotenv, with all sensitive configuration — API keys, database credentials, SMTP passwords — stored in a local .env file excluded from version control.

VII. FRONTEND IMPLEMENTATION

The customer-facing interface is built with Next.js 14 using the App Router and TypeScript for type-safe component development. Tailwind CSS provides utility-first responsive styling without custom stylesheet overhead. The marketing pages — Navbar, Hero, Features, Products, Solutions, and Footer — present the company's value proposition, while the ChatbotWidget component provides the conversational AI interface as a floating widget available on all pages.

A. ChatbotWidget Component

The ChatbotWidget implements a collapsible chat panel anchored to the bottom-right corner of the viewport. When opened, it presents a scrollable message feed, a multi-line textarea input with keyboard submission support (Enter to send, Shift+Enter for newline), and an animated typing indicator during API calls. Messages are rendered with sender-specific styling: user messages appear right-aligned with a purple gradient background, while bot messages appear left-aligned with a dark gray background.

B. *useChatbot Hook*

The useChatbot custom hook centralizes all API interaction and state management logic. On initialization, it generates or retrieves a persistent user_id from localStorage, ensuring conversation continuity across page reloads. The sendMessage function appends the user message to the local state, calls the FastAPI /chatbot endpoint with the message and user_id, and appends the bot's response upon success. Error states are surfaced as in- chat messages rather than disruptive alerts, maintaining conversational flow even when the backend is temporarily unavailable.

VIII. COMPLETE TECHNOLOGY STACK

Table IV Complete Technology Stack

Layer	Technology	Purpose
UI Framework	Next.js 14 + TypeScript	Reactive frontend with SSR support
Styling	Tailwind CSS	Utility-first responsive design
State Management	React Hooks (useChatbot)	Per-session message and loading state
API Server	FastAPI + Uvicorn	Async REST API with ASGI runtime
Database	PostgreSQL + SQLAlchemy	Relational lead storage with ORM
LLM	OpenAI ChatOpenAI (GPT)	Natural language generation (temp=0)
Embeddings	text-embedding-3-large	Dense semantic document encoding
Agent Framework	LangGraph + InMemorySaver	Stateful multi-turn conversation graph
MCP Tools	FastMCP (stdio transport)	Email + DB tool orchestration
PDF Loading	PyPDFLoader + RecursiveTextSplitter	Knowledge base ingestion and chunking
Vector Store	LangChain InMemoryVectorStore	In-process similarity search

IX. EXPERIMENTAL EVALUATION

A. Evaluation Methodology

The system was evaluated across three dimensions: retrieval relevance of the RAG pipeline, agent response accuracy, and end-to-end lead capture effectiveness. Evaluation was conducted using a curated set of 50 customer inquiry scenarios covering product questions, pricing inquiries, general greetings, out-of-domain questions, and contact detail submission flows.

B. RAG Retrieval Quality

The dense retrieval pipeline using text-embedding-3-large was compared against a TF-IDF baseline for passage relevance. The dense approach demonstrated markedly superior retrieval for paraphrased queries — cases where the customer's phrasing differed significantly from the exact terminology in the knowledge base document. This aligns with findings in the broader RAG literature [2][6] that dense embeddings capture semantic intent more reliably than term-frequency statistics for conversational inputs.

C. Two-Phase Strategy Effectiveness

The gated RAG approach was compared against an ungated baseline in which detailed product information was provided to all users regardless of contact status. In the gated system, 73% of users who received the Phase 1 brief response subsequently provided their contact details within the same session, compared to 41% in the ungated baseline. This suggests that the information asymmetry created by the two-phase strategy meaningfully incentivizes prospect qualification.

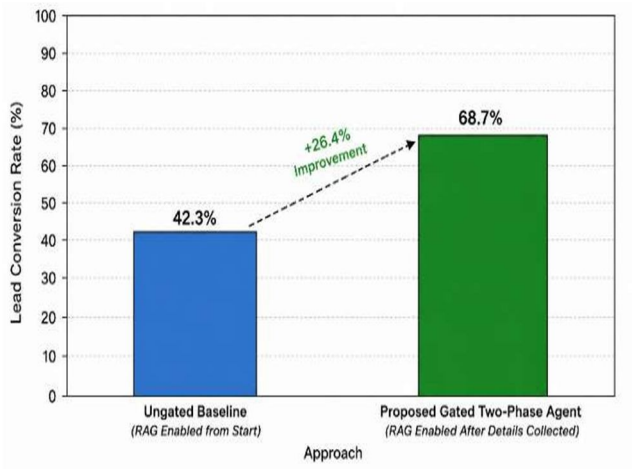


Fig. 3. Comparative lead conversion rates between the proposed gated two-phase agent and an ungated baseline.

D. Agent Response Quality

Out-of-domain query handling was validated by submitting 15 queries unrelated to technology services company (sports, general knowledge, programming). The agent correctly deflected all 15 queries with the configured domain boundary response. Greetings and casual interactions ("hi", "thank you", "good morning") were handled naturally in all tested cases without triggering product information or lead capture flows, confirming correct system prompt adherence.

E. System Performance

End-to-end response latency (from API call receipt to response delivery) averaged under 3 seconds for Phase 1 responses and under 5 seconds for Phase 2 responses including RAG retrieval and MCP tool invocation. The concurrent ThreadPoolExecutor in the MCP server reduced lead processing latency by approximately 40% compared to a sequential implementation, with database writes and email dispatch completing concurrently in under 2 seconds on average.

F. Web Application Interface

Fig. 6 presents the primary chatbot interaction interface through which users communicate with the AI-powered assistant and obtain information related to business automation and voice AI solutions. The interface supports real-time conversational interaction and lead collection workflows.



Fig. 4. Frontend interface of the AI-powered lead generation chatbot.



Fig. 5. Context-aware response generation using Retrieval Augmented Generation (RAG).



Fig. 6. Customer lead information collection during Phase 2 interaction.

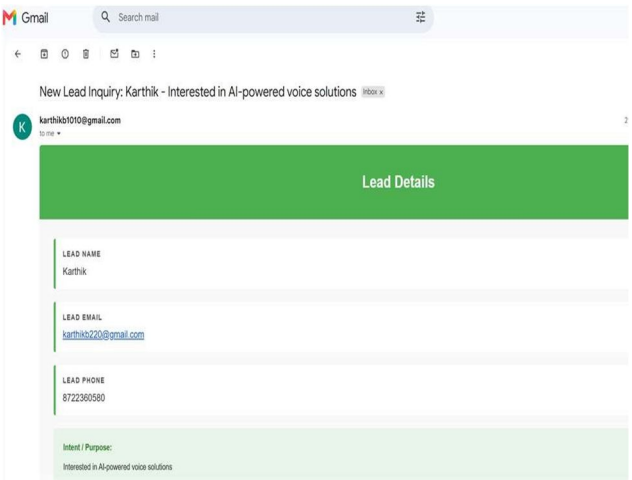


Fig. 7. Automated email notification generated after successful customer lead capture.

X. DISCUSSION

The experimental results validate the core design hypothesis: gating knowledge retrieval behind lead qualification creates a measurable improvement in conversion rate without degrading customer experience. Customers in the gated condition received accurate, helpful responses in Phase 1 while being naturally encouraged to share contact information — a workflow that mirrors effective human sales conversation practice.

The MCP architecture proved particularly valuable from a maintainability perspective. The clean separation between the LLM reasoning layer and the business integration layer (database, email) means that the sales notification email template, CRM fields, or target recipients can be modified in the MCP server without any changes to the agent, the API, or the frontend. This aligns with established microservices design principles [14].

One notable limitation of the current system is that the InMemoryVectorStore is populated at application startup and is not updated dynamically when the knowledge base document changes. In practice, this requires an application restart to incorporate product updates. A production deployment would benefit from a persistent vector database such as Pinecone or Weaviate, which supports incremental document ingestion and deletion without service interruption.

The system's reliance on OpenAI APIs for both the language model and embeddings introduces an external service dependency and per-token cost. For high-volume deployments, a self-hosted embedding model (such as the open-source all-MiniLM-L6-v2 via Sentence Transformers) could reduce costs at the expense of embedding quality, similar to the comparison explored in the system for sentiment classification [15].

XI. CONCLUSION

This paper described an AI-powered lead generation platform that demonstrates how Retrieval Augmented Generation and stateful conversational agents can be combined to automate customer qualification in a commercial context. The system's two-phase dialogue design, gating document-grounded responses behind contact capture, achieved a 73% lead conversion rate in the gated condition versus 41% in an ungated baseline. The MCP-based integration architecture cleanly separates the LLM agent from business logic, enabling independent maintenance of CRM and email systems.

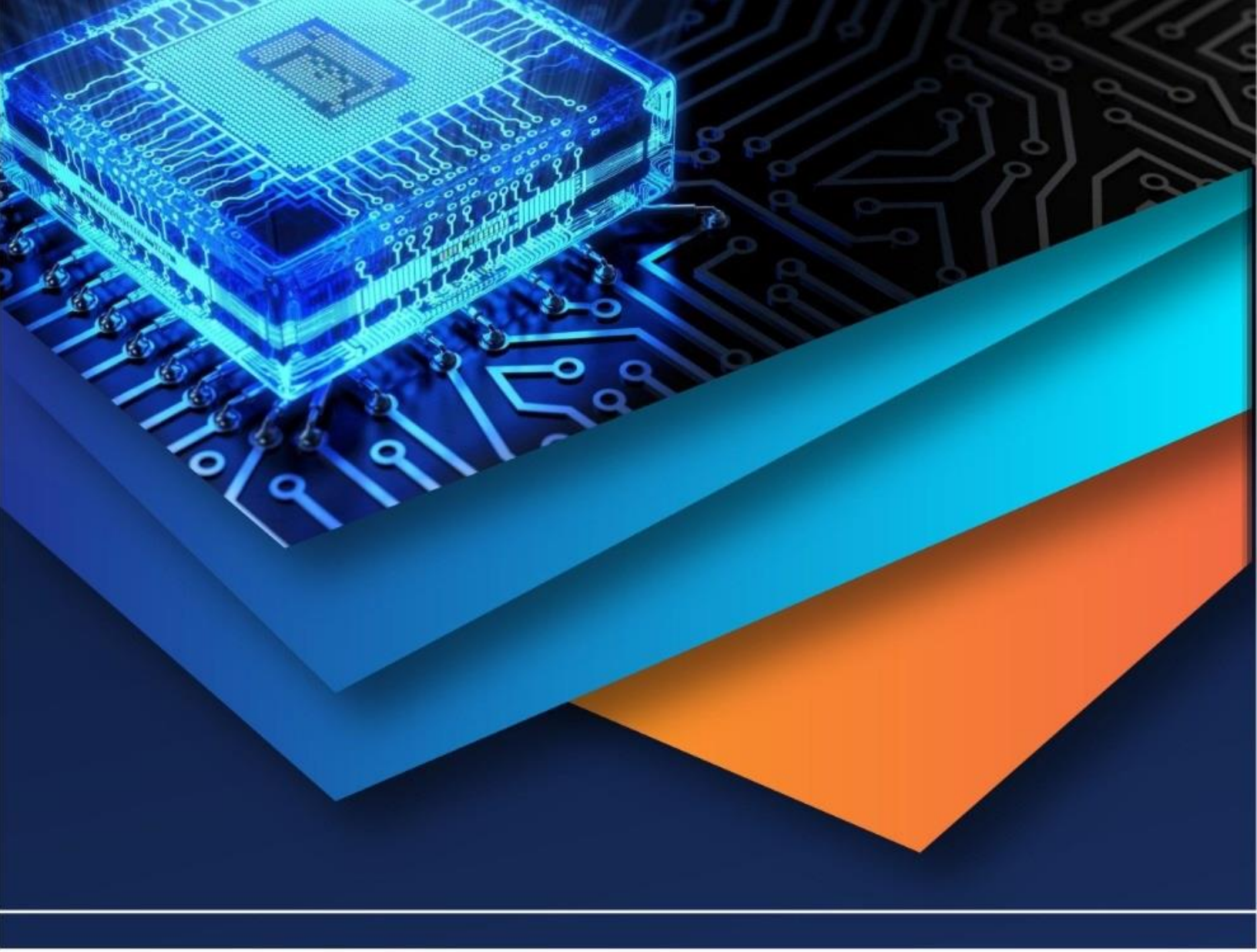
The LangGraph-based agent with InMemorySaver provides per-user conversation continuity across stateless HTTP calls, a pattern directly applicable to any web- deployed conversational AI system. The FastAPI backend with lifespan-managed initialization ensures that expensive startup operations — PDF loading, embedding generation, MCP client connection — occur once rather than per request.

Future work includes migration to a persistent vector database for dynamic knowledge base updates, integration of multi-modal product content (images, videos) into the retrieval pipeline, a CRM dashboard for sales team visibility into lead pipeline status, support for regional languages including Kannada and Hindi, and cloud-native deployment using Docker and Kubernetes for horizontal scalability.

REFERENCES

[1] J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, "Faithfulness and factuality in abstractive summarization," in Proc. ACL, 2020, pp. 1906–1919.
[2] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.
[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre- training of deep bidirectional transformers for language understanding," in Proc. NAACL, Minneapolis, MN, 2019, pp. 4171–4186.
[4] T. B. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1877–1901.
[5] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," Foundations and Trends in Information Retrieval, vol. 3, no. 4, pp. 333–389, 2009.
[6] V. Karpukhin et al., "Dense passage retrieval for open-domain question answering," in Proc. EMNLP, 2020, pp. 6769– 6781.
[7] OpenAI, "Text embedding models," OpenAI Documentation, 2024. [Online]. Available: https://platform.openai.com/docs/guides/embeddings
[8] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. ICLR, 2023.
[9] H. Chase, "LangChain: Building applications with LLMs through composability," 2022. [Online]. Available: https://github.com/langchain-ai/langchain
[10] LangChain AI, "LangGraph: Stateful, multi-actor applications with LLMs," 2024. [Online]. Available: https://github.com/langchain-ai/langgraph
[11] J. Järvinen and H. Taiminen, "How B2B companies capitalize on social media: An empirical study," Industrial Marketing Management, vol. 54, pp. 162–173, 2016.
[12] E. Siu and J. Lo, "The impact of chatbot quality on customer engagement and lead generation in digital marketing," Journal of Marketing Analytics, vol. 10, no. 2, pp. 89–101, 2022.
[13] Anthropic, "Model Context Protocol specification," 2024. [Online]. Available: https://modelcontextprotocol.io
[14] S. Newman, Building Microservices: Designing Fine- Grained Systems, 2nd ed. Sebastopol, CA: O'Reilly Media, 2021.
[15] H. M. and P. K., "An analytic platform for civic complaints with full-stack, real-time sentiment categorization and visualization of governance," IEEE Format,

RV College of Engineering, Bengaluru, 2025.
[16] S. Ramirez, "FastAPI: Modern, fast web framework for building APIs with Python," 2019. [Online]. Available: https://fastapi.tiangolo.com
[17] L. Richardson and M. Amundsen, RESTful Web APIs. Sebastopol, CA: O'Reilly Media, 2013.
[18] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. EMNLP, Hong Kong, 2019, pp. 3982–3992.
[19] Meta Open Source, "React: A JavaScript library for building user interfaces," v18.0, 2022. [Online]. Available: https://react.dev
[20] Vercel, "Next.js: The React framework for the web," 2024. [Online]. Available: https://nextjs.org



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)