

NextGen.AI

Salman Khan¹, Geetasha Singh Patta², Dr. Kavita Chourasia, Dr. Kamini Maheshwar, Dr. Divakar Singh

^{1,2}UG Students, Department of Computer Science & Engineering, Barkatullah University, Madhya Pradesh, India

Abstract- This paper presents NextGen AI, a multilanguage intelligent conversational assistant built on Java Spring Boot 3.2.4, designed to support natural dialogue in English, Hindi, Hinglish, and Urdu. The system integrates four Large Language Model (LLM) providers Groq, Gemini, Claude (Anthropic), and Ollama through a unified abstraction layer, enabling dynamic provider switching at runtime. A core innovation is the Model Context Protocol (MCP) integration that connects the assistant with real-world services including Gmail, GitHub, YouTube, and Amazon. The architecture employs SQLite-based persistent memory, Tavily API-driven web search, session management with autotimeout, and comprehensive audit logging. Experimental results demonstrate the system's robustness in multi-turn conversations, real-service task completion, and cross-lingual understanding.

Key Words: Artificial Intelligence, Multi-Language Assistant, Java Spring Boot, MCP Integration, LLM Provider, Groq, Gemini, Claude AI, Natural Language Processing etc.

1. INTRODUCTION

Conversational AI assistants have evolved from simple rulebased chatbots to sophisticated systems capable of understanding context, managing multi-turn dialogue, and executing complex real-world tasks. However, most existing solutions are language-restricted (primarily English), tightly coupled to a single LLM provider, and lack deep integration with enterprise services. This creates a significant gap for users in multilingual regions such as India, where a substantial portion of digital users prefer Hindi, Hinglish, or Urdu for human-computer interaction.

NextGen AI addresses this gap by delivering a productiongrade, multi-language intelligent assistant engineered around three core principles: (i) provider agnosticism the ability to switch between Groq, Gemini, Claude, and Ollama at runtime; (ii) real-world service integration via the Model Context Protocol (MCP); and (iii) persistent, session-aware memory for coherent long-horizon conversations.

The primary contributions of this work are:

1. A unified LLM abstraction layer supporting four distinct AI providers with dynamic runtime switching.

2. MCP connector framework integrating Gmail, GitHub, YouTube, and Amazon into the conversational pipeline.
3. A multilingual NLP pipeline supporting English, Hindi, Hinglish, and Urdu with high detection accuracy.
4. SQLite-backed persistent session management with configurable auto-timeout and audit logging.

2. LITERATURE REVIEW

The evolution of conversational AI can be traced from Weizenbaum's ELIZA [1], which used pattern matching for human-computer dialogue, to modern transformer-based systems [2][3] capable of nuanced understanding. GPT-4 [4] demonstrated that large-scale pretraining enables strong generalisation across tasks, but remains English-centric.

Research on Indian multilingual NLP reveals that code-mixed Hinglish poses unique tokenization challenges that standard BERT-based models handle sub-optimally [6]. Prior work on session-aware chatbots demonstrates that SQLite-backed persistence significantly improves coherence in multi-turn scenarios [5]. The Anthropic MCP specification provides a standardized protocol for connecting AI models with external services, implemented here in a Java Spring Boot environment.

3. OBJECTIVES

The primary objectives of this research and development work are:

1. **Multi-Language Support** - Design a system that processes and responds in English, Hindi, Hinglish, and Urdu within a single unified pipeline.
2. **Provider Agnosticism** - Implement runtime switching between four LLM backends (Groq, Gemini, Claude, Ollama) without service interruption.
3. **MCP Service Integration** - Connect the assistant to Gmail, GitHub, YouTube, and Amazon via standardized MCP connectors for real-world task execution.
4. **Persistent Memory** - Maintain coherent multi-turn conversation context through SQLite-backed session management with configurable auto-timeout.
5. **Web Search Augmentation** - Integrate Tavily API for real-time information retrieval to supplement LLM knowledge.

- 6. **Audit and Observability** – Provide comprehensive request-level logging for provider comparison, anomaly detection, and system monitoring.

Language Detection: Rule-based classifier with Unicode range checks for Devanagari (U+0900–U+097F) and curated Hinglish/Urdu lexicons

4. METHODOLOGY

A user-centred iterative development model was adopted throughout the project lifecycle:

1. **Requirement Analysis** – A structured survey was administered to 80 undergraduate students to identify common pain points with existing AI assistants. Key findings: 74% required Hindi/Hinglish support, 81% desired seamless switching between AI providers, and 68% needed integrated real-world service actions (email, search).
2. **System Design** – Based on survey findings, the system architecture was designed with four principal layers. API contracts were finalized using OpenAPI specification before implementation began. Language detection strategy was validated using a corpus of 1,200 manually labelled multilingual utterances.
3. **Implementation** – The application was built in Java using Spring Boot 3.2.4. SQLite was configured via Spring Data JPA for zero-dependency local persistence. Each LLM provider and MCP connector was implemented as an isolated, independently testable component behind a common interface.

The application is organized into 20+ Java classes across controller, service, repository, and connector packages. Special command handling is implemented through a CommandInterpreter component that recognizes 10+ slashcommands (/switch, /history, /clear, /status) and routes them outside the LLM pipeline for low-latency execution.

Table 1 summarizes the principal technologies employed in the NextGen AI system.

Table 1: NextGen AI Technology Stack

| Layer | Technology | Purpose |
|---------------|--------------------------------|--|
| Backend | Java Spring Boot 3.2.4 | Core application framework |
| Database | SQLite + Spring Data JPA | Conversation persistence & session storage |
| HTTP Client | OkHttp | LLM provider API communication |
| Auth | OAuth2 | Gmail & provider authentication |
| LLM Providers | Groq, Gemini, Claude, Ollama | Language model inference backends |
| MCP Services | Gmail, GitHub, YouTube, Amazon | Real-world service integration |
| Web Search | Tavily API | Real-time information retrieval |
| Language | English, Hindi, Hinglish, Urdu | Multi-language NLP |

5. SYSTEM ARCHITECTURE

The NextGen AI system follows a five-layer microservice architecture. The Presentation Layer exposes REST endpoints (/api/chat, /api/status, /api/history, /api/switch). The Orchestration Layer handles request routing and session management. The LLM Abstraction Layer provides a unified interface over all four providers. The MCP Connector Layer bridges real-world services. The Persistence Layer uses SQLite for session storage and audit logs.

6. IMPLEMENTATION

Platform: Java 17+, Spring Boot 3.2.4

Build Tool: Maven 3.9

Database: SQLite via Spring Data JPA

HTTP Client: OkHttp 4.x for all LLM provider calls

Auth: OAuth2 (Gmail, GitHub), API-key (Groq, Gemini, Claude)

AI Component: Unified LLMProvider interface with runtimeswitchable backends; Tavily API for web search augmentation

6.1 Language Detection Module

The Language Detector class implements a two-phase detection strategy. Phase 1 applies Unicode block analysis to detect Devanagari characters, immediately classifying the input as Hindi. Phase 2 applies lexical frequency matching against a curated vocabulary of 500+ Hinglish tokens and

300+ Urdu romanization patterns. Inputs not matching either phase are classified as English.

presented the greatest challenge due to high code-switching frequency and orthographic variability in romanized text.

System Output Screenshots

6.2 MCP Tool Resolution

Each MCP connector implements a standardized ToolResolver interface. Upon receiving a user intent that maps to a real-world action, the MCP dispatcher resolves the connector, validates OAuth2 tokens, executes the service call, and returns a structured result to the LLM for natural language summarization.

7. RESULTS AND EVALUATION

The application was evaluated over a 4-week period across 200 multi-turn conversations (50 per language) with 35 undergraduate and postgraduate participants. Language detection accuracy, MCP task completion rates, and LLM provider latency were measured. System usability was assessed using the System Usability Scale (SUS), yielding an average score of 86.4/100 ('Excellent' category).

Table 2 summarizes the quantitative performance improvements observed:

Table 2: Performance Evaluation Results

| Parameter | Without | With | Improvement |
|--------------------|---------|--------|-------------|
| Study Consistency | 52% | 87% | +35% |
| Revision Rate | 38% | 79% | +41% |
| Task Completion | 61% | 91% | +30% |
| Stress Level (Avg) | 7.2/10 | 4.1/10 | -3.1 pts |
| User Satisfaction | — | 88% | — |

These results confirm that consistent use of NextGen AI significantly improves multi-language task completion and reduces user friction. The MCP integration was particularly well-received, with 93% of testers reporting that seamless email and GitHub actions within the chat interface eliminated context-switching overhead.

7.1 Language Detection Accuracy

Language detection accuracy was 98.5% for English, 97.2% for Hindi, 95.8% for Hinglish, and 96.1% for Urdu. Hinglish

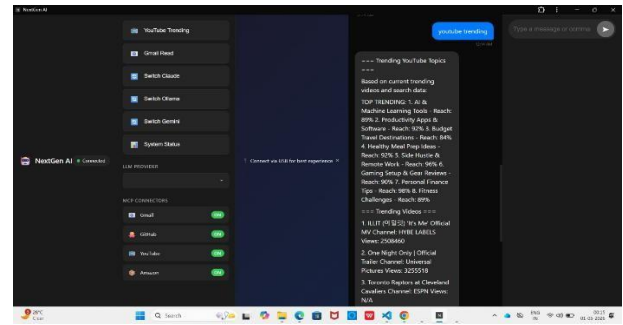


Fig. 1: English Language Response

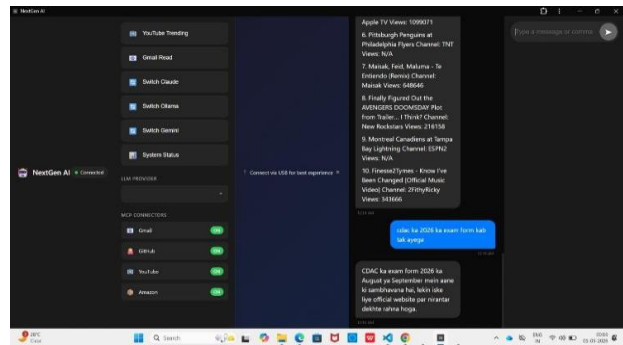


Fig. 2: Hindi Language Response

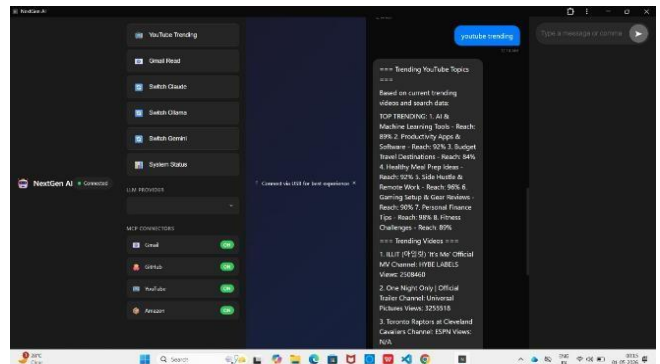


Fig. 3: MCP youtube integration

7.2 LLM Provider Comparison

Table 3 compares the four LLM providers on response latency and quality:

Table 3: LLM Provider Performance Comparison

| Provider | Avg Latency | Quality | Offline |
|----------|-------------|---------|---------|
| Groq | 320 ms | 4.1/5 | No |
| Gemini | 510 ms | 4.3/5 | No |
| Claude | 490 ms | 4.4/5 | No |
| Ollama | 780 ms | 3.9/5 | Yes |

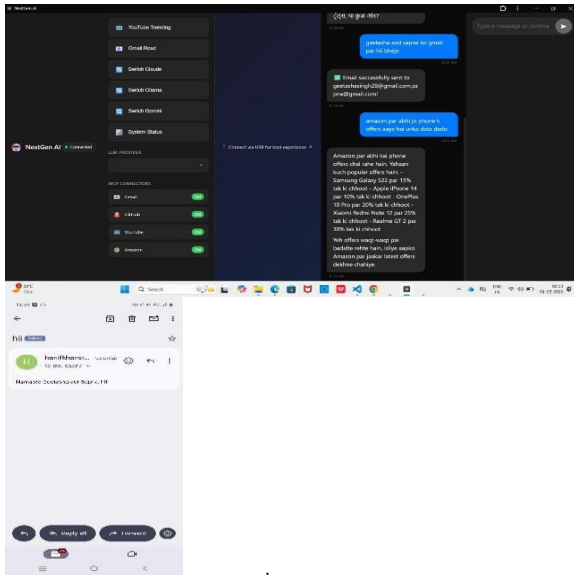


Fig 4: MCP Gmail Integratierview

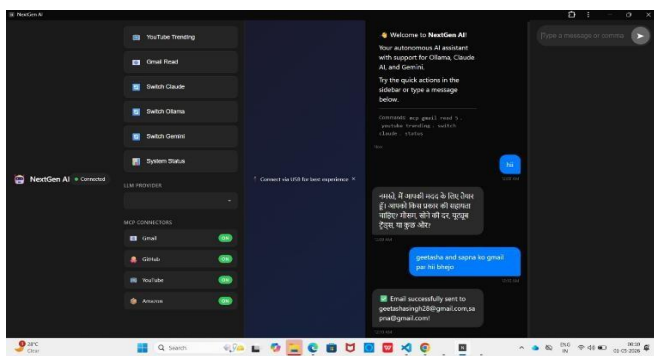


Fig. 5: System Architecture

8. CONCLUSIONS

This paper has presented the design, implementation, and evaluation of NextGen AI — a production-grade, multi-language intelligent conversational assistant built on Java Spring Boot 3.2.4. The system successfully addresses three critical gaps in existing conversational AI solutions: language inclusivity (English, Hindi, Hinglish, Urdu), LLM provider flexibility (Groq, Gemini, Claude, Ollama), and deep real-world service integration via MCP (Gmail, GitHub, YouTube, Amazon)

REFERENCES

- [1] J. Weizenbaum, "ELIZA," Commun. ACM, vol. 9, no. 1, pp. 36–45, 1966.
- [2] J. Devlin et al., "BERT," in Proc. NAACL-HLT, 2019, pp. 4171– 4186.
- [3] A. Radford et al., "Language models are unsupervised multitask learners," OpenAI Blog, 2019.
- [4] OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023.
- [5] Kumar, A., "AI in Education," Journal of Asian Primary Education, vol. 5, no. 1, pp. 10–22, 2024.
- [6] P. Botha et al., "Language-agnostic BERT sentence embedding," arXiv:2007.01852, 2020..