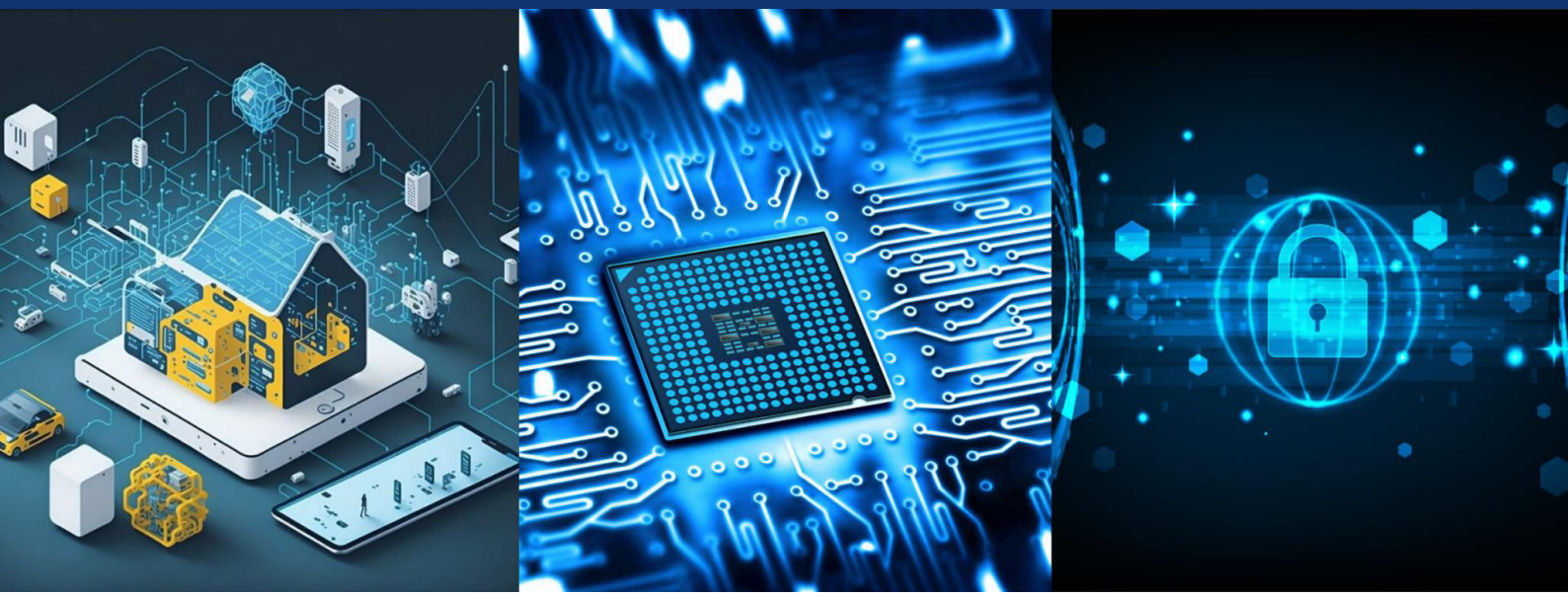
IJIRCCE

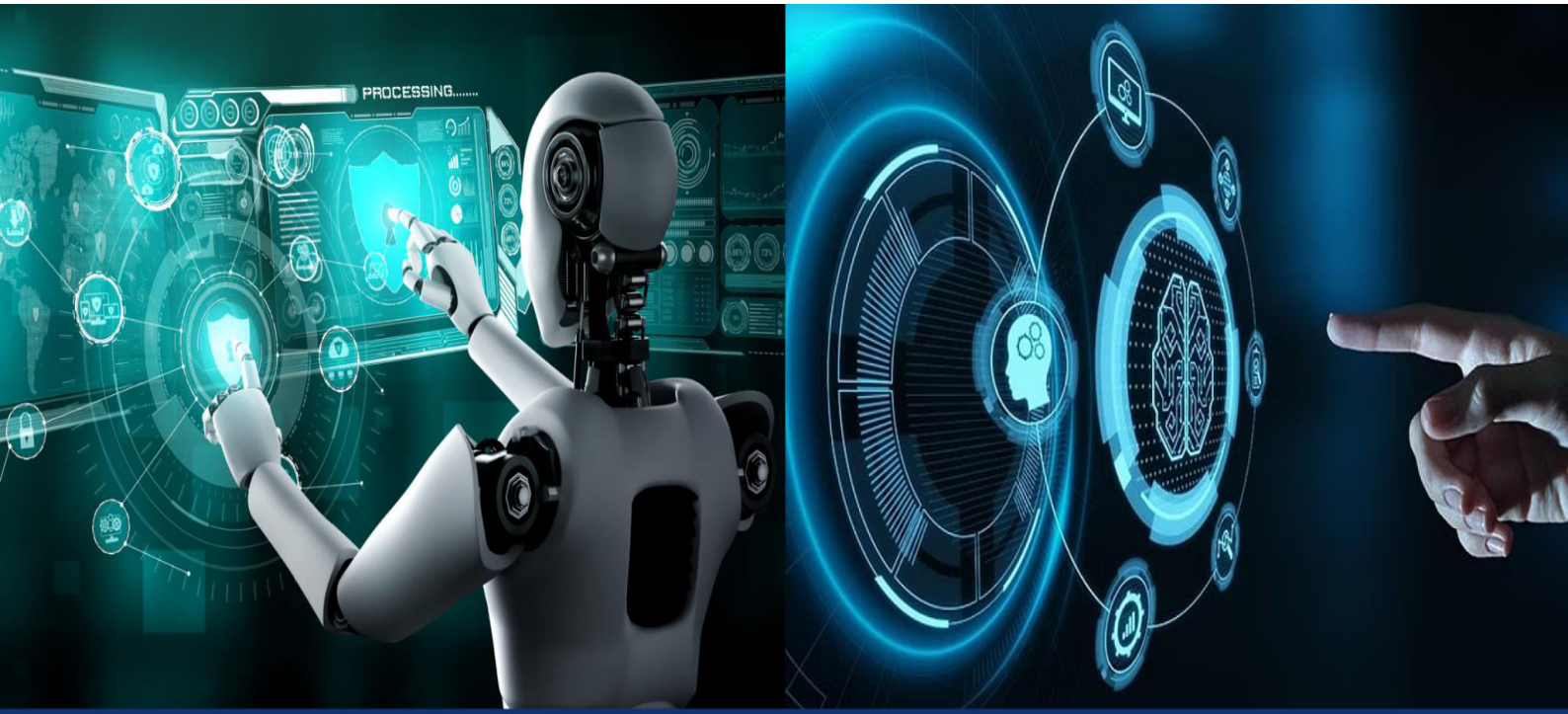
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 5, May 2026

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405071

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

JITD AI, a Fully Offline RAG-based University Assistant System using Local LLM with Ollama

Spandana Mahadevappa Kandagal[1],Vasistha C V[1], Sudeep Sagar[1], Rithin P Vali[1], Dr. Latha BM[2], Manjula P[3]

UG Students, Dept. of CSE, Jain Institute of Technology Davanagere, Karnataka, India[1]

Head and Professor, Dept. of CSE, Jain Institute of Technology Davanagere, Karnataka, India[2]

Assistant Professor, Dept. of CSE, Jain Institute of Technology Davanagere, Karnataka, India[3]

ABSTRACT: Large Language Models (LLMs) are widely used in applications such as chatbots, code generation, and automated assistance systems. However, most existing LLMs rely on cloud-based platforms, creating challenges related to data privacy, internet dependency, latency, and cost. In secure or low-connectivity environments, these systems may become less efficient. This project aims to develop a fully offline Local LLM-based system that can generate responses, execute code, and detect errors directly on a local machine. The system integrates locally deployed LLMs with Natural Language Processing (NLP) and runtime

KEYWORDS: Local Large Language Models (LLMs); Natural Language Processing (NLP); Data Privacy and Security; Local Deployment; Ollama; AI Assistant.

I. INTRODUCTION

Large Language Models (LLMs) are advanced Artificial Intelligence systems capable of understanding natural language, generating human-like responses, and assisting users in tasks such as coding, content generation, and problem solving. Most modern LLM systems are cloud-based and require continuous internet connectivity to process user requests. Although cloud-based systems provide powerful computational capabilities, they also introduce challenges such as data privacy risks, higher latency, dependency on external servers, and increased operational costs. In many real-world scenarios, especially in educational institutions, research environments, and secure organizations, users require AI systems that can operate offline without sharing sensitive information over the internet.The proposed Local LLM system aims to provide an intelligent offline AI assistant capable of natural language interaction, local code generation, runtime execution, and automatic error detection. The system integrates locally deployed LLMs with Natural Language Processing (NLP) and execution frameworks to perform tasks directly on the local machine without external communication. By utilizing lightweight deployment and optimization techniques, the system improves privacy, reduces latency, and ensures reliable offline AI assistance while efficiently utilizing available hardware resources.

II. RELATED WORK

The authors assessed the effectiveness of local Large Language Model deployment using frameworks such as Ollama, GPT4All, [3] and Llamafile with models like Mixtral, [4] Gemma 2, and LLaMA 3. The study highlighted that local deployment improves data privacy, autonomy, and offline accessibility, but larger models require higher computational resources and may increase latency on low-end devices. Researchers also integrated local LLMs into intelligent log analysis systems [5], where Ollama, database queries, and long-chain splitting techniques were used to process large log files efficiently while maintaining data privacy.

The authors presented OllamaRouter, an adaptive load balancing architecture [6] for distributed LLM inference, improving throughput and reducing latency. Researchers also explored privacy-preserving AI applications [7] using optimization techniques such as quantization, pruning, and memory management to support offline model execution.

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405071

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Another study proposed a secure Natural Language-to-SQL system [6] with policy-aware middleware for safe query execution. Furthermore, optimization methods including prompt engineering and runtime management [7] were suggested to improve local LLM performance, achieving better resource utilization and near cloud-level accuracy.

III. PROPOSED ALGORITHM

The proposed methodology explains the working process of the Local LLM-based AI system using Ollama.[8] The system provides offline AI assistance for general queries and code-related tasks by integrating frontend interaction, backend processing, local LLM response generation, code execution, and automatic error handling. The methodology ensures secure, efficient, and real-time AI interaction without internet dependency.

1. USER INTERFACE (INPUT LAYER)

The system begins with a user-friendly interface developed using HTML, CSS, and JavaScript, where the user enters a prompt or query. This interface acts as the primary interaction point between the user and the system. It supports both general queries and code-related requests, ensuring flexibility in usage. The interface is designed to be simple, responsive, and efficient, allowing users to interact with the system seamlessly. By capturing user input accurately, it ensures that the request is properly forwarded for further processing.

2. Fetch API Request (Client-Server Communication)

Once the user submits a query, a Fetch API request[9] is sent from the frontend to the backend server built using Node.js and Express.js. This step establishes communication between the client and server, ensuring that user input is transmitted securely and efficiently. The backend receives the request and prepares it for further processing.

3. Request Type Detection (Decision Layer)

After receiving the request, the system analyzes the input to determine its type-whether it is a general query or a code-related request. This decision-making process is essential for routing the request to the appropriate workflow. If the query is general, it is sent directly to the language model for response generation. If it is a code request, the system follows a different pipeline involving code generation and execution. This intelligent classification improves efficiency and ensures accurate handling of different types of inputs.

4. General Query Processing (LLM Response Generation)

For normal queries, the system sends the input to the local Large Language Model using Ollama. The model processes the query and generates a relevant and human-like response. Since the model runs locally, there is no need for internet connectivity, ensuring privacy and low latency. The generated response is then sent back to the frontend and displayed to the user as output.

5. Code Generation using LLM

If the request is identified as a code-related query, the system forwards it to the local LLM to generate appropriate code. The model understands the user's requirements and produces executable code in the desired programming language. This step transforms natural language input into functional code, enabling automation and problem-solving capabilities within the system.

6. Temporary File Storage

The generated code is stored in a temporary file within the system. This step is necessary to prepare the code for execution. By storing the code locally, the system ensures better control, security, and efficient handling of execution processes. It also allows the system to manage multiple requests and maintain execution logs if required.

7. Code Execution (OS-Level Processing)

The stored code is executed using operating system-level processes. This allows the system to run scripts, programs, or commands directly on the local machine. The execution process is monitored carefully to capture outputs and detect any errors. This capability makes the system more powerful compared to traditional AI tools, as it not only generates but also executes code in real time.

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405071

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

8. Error Detection and Handling

After execution, the system checks whether any errors have occurred. If an error is detected, the error details along with the original code are sent back to the LLM. This enables the system to analyze the issue and regenerate corrected code. This loop continues until the code executes successfully.

9. Response Generation and Output Display

Once the code executes successfully or a valid response is generated, the final output is sent back to the frontend. The result is displayed to the user in a clear and understandable format. This ensures that users receive accurate responses, whether it is general information or execution output.

10. Continuous Feedback Loop (Self-Healing System)

The system incorporates a feedback loop where errors are automatically corrected through interaction with the LLM. This makes the system adaptive and intelligent over time.[10] By continuously refining outputs based on execution results, the system ensures high performance, reliability, and user satisfaction.

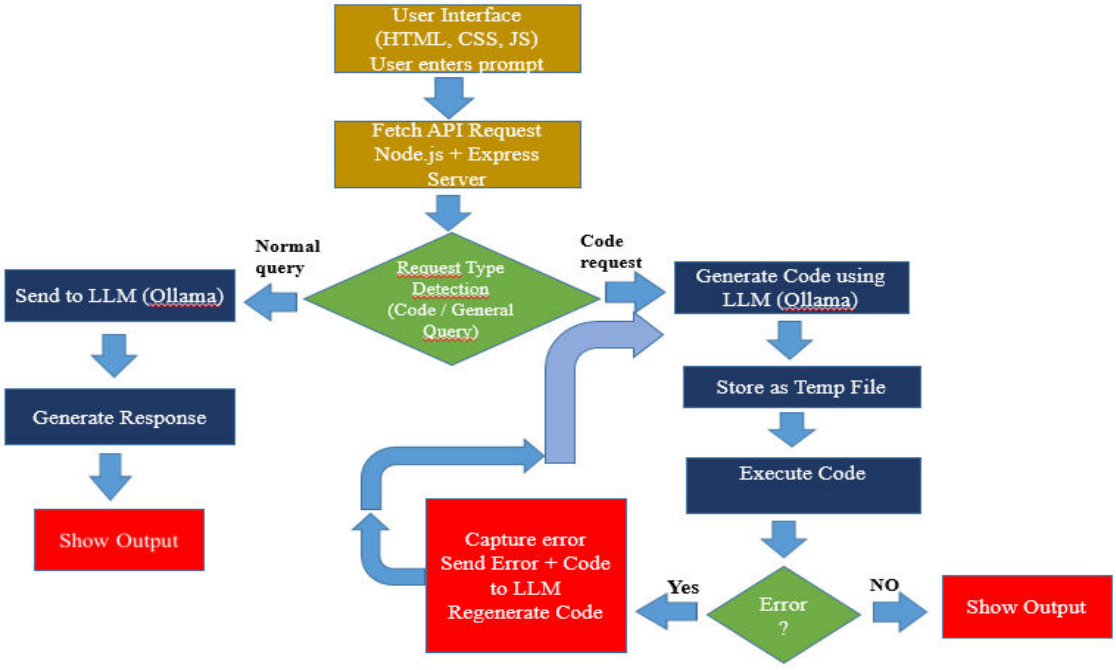


Figure1: Proposed Methodology Architecture

IV. PSEUDO CODE

Step 1: Initialize the Local LLM model using the Ollama framework.
Step 2: Accept the user query through the chatbot interface.
Step 3: Preprocess the user query using NLP techniques.

```
if (query contains unnecessary symbols or invalid input)
        Clean and normalize the query
else
        Pass the processed query to the Local LLM
end
```

Step 4: Generate the response using the Local LLM model.
Step 5: Check whether the generated output contains executable code.

```
if (output contains code)
        Send the code to the local execution environment
```

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405071

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

else

Display the generated response to the user

end

Step 6: Execute the generated code locally.

Step 7: Detect syntax or runtime errors during execution.

if (error detected)

Generate corrected code suggestion

Re-execute the corrected code

else

Display successful execution output

end

Step 8: Store execution history and generated responses locally for future interaction.

Step 9: Continue processing user queries until the session ends.

Step 10: End.

V. SIMULATION RESULTS

The experimental study implemented the proposed Local LLM system using locally deployed language models through the Ollama framework [11] integrated with NLP and runtime execution modules. [12] The system was tested using chatbot interactions, code generation, and automatic error correction tasks to analyze response time, execution accuracy, and offline usability.The system successfully generated conversational responses and executable code without internet connectivity. It effectively detected syntax and runtime errors, providing corrected code suggestions. Results showed improved privacy, secure local processing, reduced cloud dependency, and reliable performance, making the system suitable for educational, research, and privacy-sensitive environments.

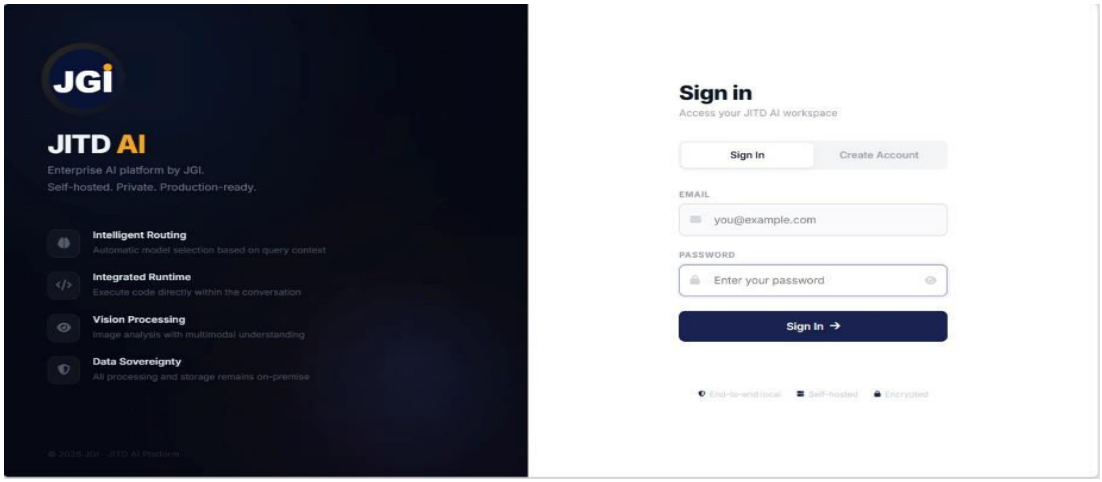


Fig 1: User Login Page

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405071

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

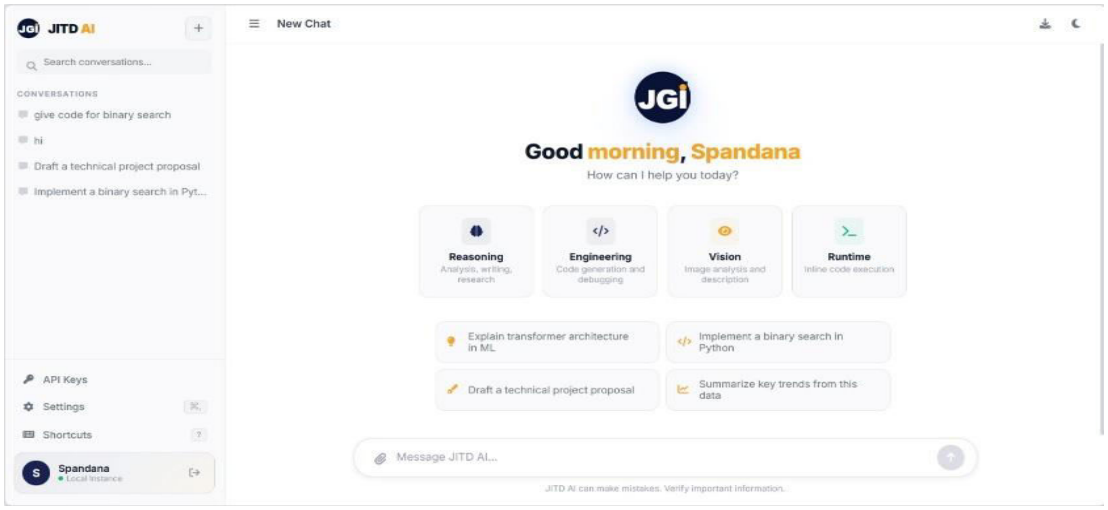


Fig 2: Dashboard Interface Page

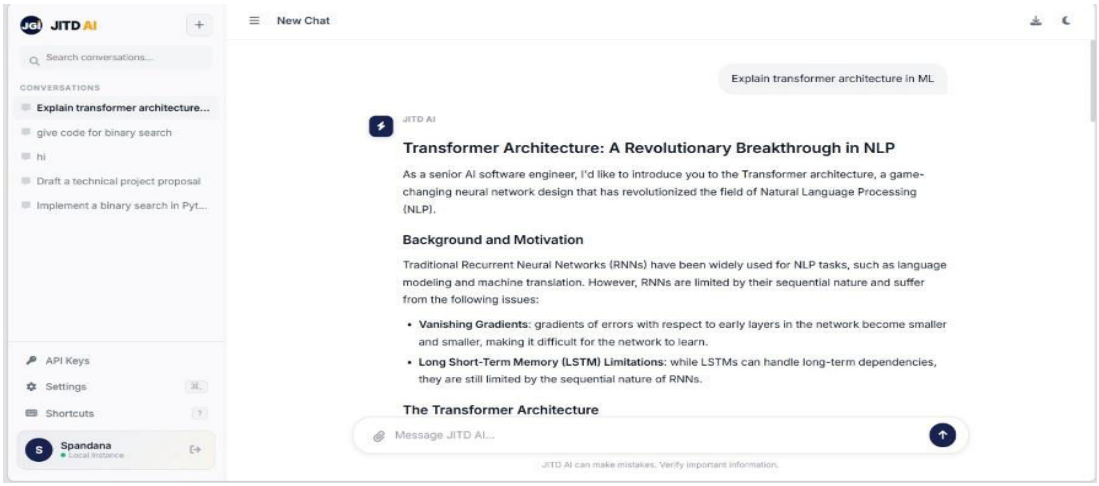


Fig 3: Chatbot Response Output Page

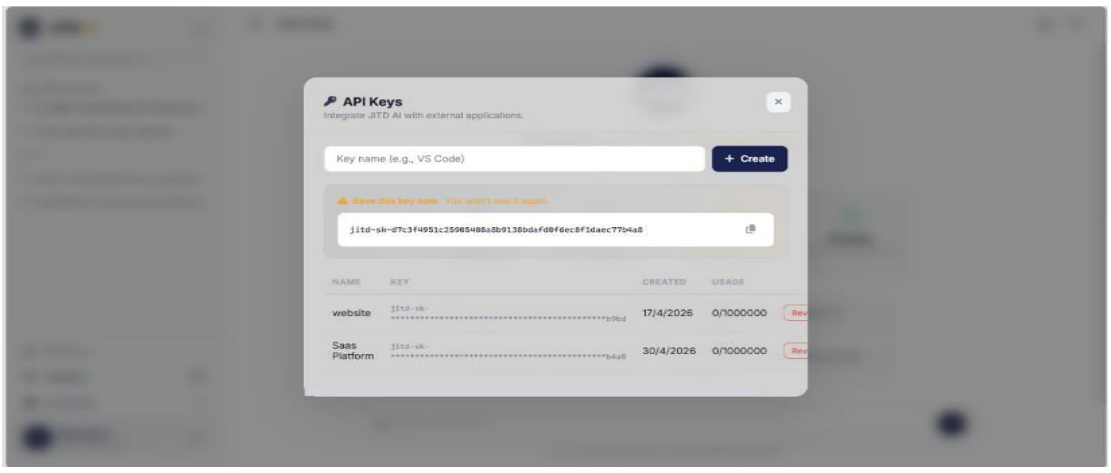


Fig 4: API Key Generation and Management

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405071

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

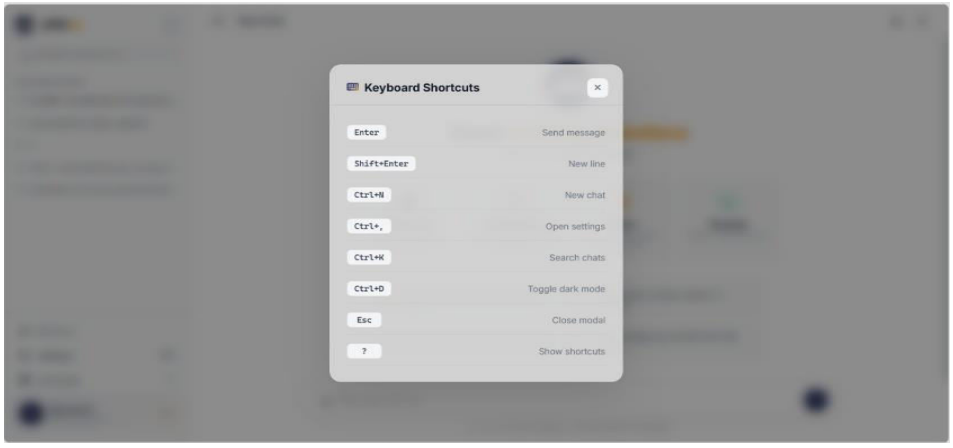


Fig 5: Keyboard Shortcuts Panel

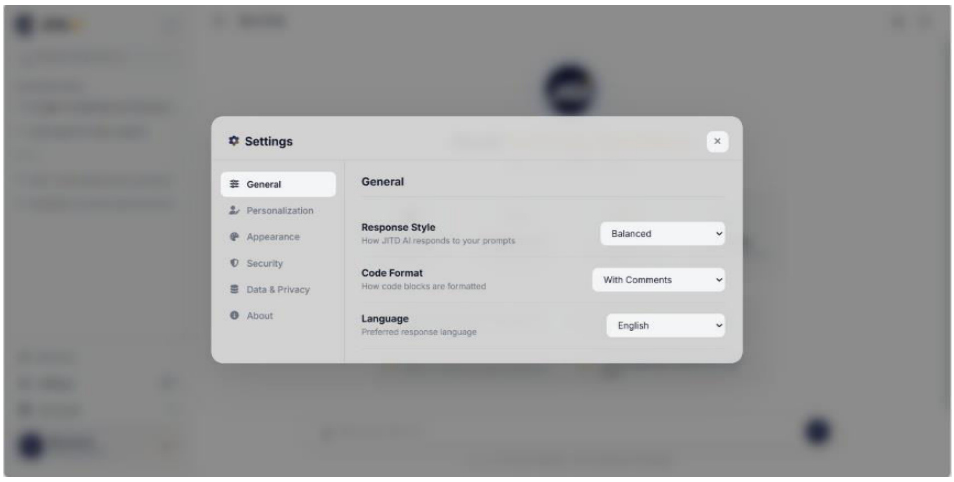


Fig 6: System Settings Configurations Page

VI. CONCLUSION AND FUTURE WORK

The experimental results showed that the proposed Local LLM system performs efficiently in terms of offline AI assistance, chatbot interaction, and local code execution. The system provides secure and reliable processing without depending on cloud-based services and ensures improved data privacy and reduced latency. The integration of Local LLMs, Natural Language Processing, and error handling mechanisms enables accurate response generation and intelligent user interaction within a local environment.As the performance of the proposed system was analyzed on moderate hardware configurations, future work can focus on improving scalability and computational efficiency using advanced optimization techniques. Additional features such as Retrieval-Augmented Generation (RAG), multimodal support, and distributed local inference can also be implemented to enhance the system performance and usability.

REFERENCES

1. Kechaoui, T., Ouhab, M. W., Djamaa, B., & Senouci, M. R. (2025, April). Locally-deployed Open-source LLMs for Code Generation: Promises and Challenges. In *2025 7th International Conference on Pattern Analysis and Intelligent Systems (PAIS)* (pp. 1-6). IEEE.
2. Umesh, R., & Kumar, P. (2025, October). AI-Powered Offline Voice Assistant for Rural Communities. In *2025 7th International Conference on Innovative Data Communication Technologies and Application (ICIDCA)* (pp. 1468-1474). IEEE.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

3. Anand, Yuvanesh, et al. "GPT4All: An ecosystem of open source compressed language models." *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*. 2023.
4. Jiang, Albert Q., Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot et al. "Mixtral of experts." *arXiv preprint arXiv:2401.04088* (2024).
5. Wang, Yongheng, Wensheng Gan, and S. Yu Philip. "AI-Driven Log Analysis: Advances and Challenges." In *2025 IEEE International Conference on Big Data (BigData)*, pp. 7728-7743. IEEE, 2025.
6. Alam, M.S., Javed, Q., Akbar, M., Rehman, M.R.U. and Anwer, M.B., 2004, June. Adaptive load balancing architecture for snort. In *2004 International Networking and Communication Conference* (pp. 48-52). IEEE.
7. Ogrezeanu, Iulian, Anamaria Vizitiu, Costin Ciușdel, Andrei Puiu, Simona Coman, Cristian Boldișor, Alina Itu et al. "Privacy-preserving and explainable AI in industrial applications." *Applied Sciences* 12, no. 13 (2022): 6395.
8. Dresselhaus, Nicole. "Case Study: Local LLM-Based NER with n8n and Ollama." (2025).
9. Simpson, J. (2023). Fetching Data, APIs, and Promises. In *How JavaScript Works: Master the Basics of JavaScript and Modern Web App Development* (pp. 223-258). Berkeley, CA: Apress.
10. Hayes-Roth, Barbara. "An architecture for adaptive intelligent systems." *Artificial intelligence* 72, no. 1-2 (1995): 329-365.
11. Marcondes, Francisco S., et al. "Using ollama." *Natural Language Analytics with Generative Large-Language Models: A Practical Approach with Ollama and Open-Source LLMs*. Cham: Springer Nature Switzerland, 2025. 23-35.
12. Long, Sifan, et al. "A survey on intelligent network operations and performance optimization based on large language models." *IEEE Communications Surveys & Tutorials* 27.6 (2025): 3915-3949.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details