



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 Issue: V Month of publication: May 2025

DOI: <https://doi.org/10.22214/ijraset.2025.70349>

www.ijraset.com

Call:  08813907089

E-mail ID: ijraset@gmail.com

Intelligent Chatbot Application Using Flutter for Cross- Platform Integration

Mrs. K.Prathyusha¹, B. Kranthi Kumar Varma², B. Vishal³, B. Varshitha⁴, B. Uday⁵

¹Assistant Professor, ^{2, 3, 4, 5}Student, Teegala Krishna Reddy Engineering College

Abstract: *This project focuses on the development of a chatbot application using the Flutter framework, designed to provide a seamless conversational experience across multiple platforms, including Android, iOS, and the web. The primary objective is to demonstrate the capability of Flutter in building an intelligent chat bot with a consistent user interface and high performance. The chat bot utilizes Flutter's rich widget library to create an interactive and intuitive chat interface. It incorporates predefined conversational logic with potential integration of natural language processing (NLP) capabilities through third-party APIs to enhance response accuracy and adaptability.*

I. INTRODUCTION

In recent years, the integration of artificial intelligence (AI) with user-centric applications has revolutionized human-computer interaction, especially through conversational agents like chatbots. While cloud-based solutions such as OpenAI's ChatGPT and Google Bard have gained massive popularity, they often come with limitations such as data privacy concerns, recurring costs, and dependence on internet connectivity. This research focuses on the development of a Flutter-based offline AI chatbot application for Windows desktop, designed to operate without reliance on cloud APIs, thereby offering a low-cost, privacy-conscious alternative for users. The proposed application is developed using Flutter, an open-source UI toolkit by Google, which allows for building visually appealing, cross-platform desktop applications. The chatbot utilizes Ollama, a locally hosted inference server that enables seamless interaction with large language models (LLMs) such as LLaMA and Mistral. By leveraging HTTP communication with the Ollama backend, the application enables real-time natural language processing and response generation directly on the user's machine. A key objective of this project is to design a modern, user-friendly interface inspired by popular AI assistants. Features include a fixed-bottom input box, dynamic chat flow with upward scrolling messages, and a contextual chat history panel that updates in real-time and allows users to start new conversations. The offline architecture not only reduces the dependency on high-speed internet but also provides enhanced control over user data and system performance, making it suitable for educational, research, and low-resource environments.

II. LITERATURE REVIEW

The emergence of intelligent conversational agents has significantly transformed the digital interaction landscape, making chatbots a central feature in various domains, including customer service, education, and personal assistance. This literature review examines existing research and technologies related to AI chatbots, offline AI systems, and cross-platform development frameworks to contextualize the current project.

A. AI-Powered Chatbots

AI chatbots have evolved from rule-based systems to sophisticated natural language processing (NLP) agents powered by large language models (LLMs). Studies such as those by [Jurafsky & Martin, 2020] highlight how LLMs like GPT and BERT have enabled more natural and context-aware conversations. Cloud-based services like OpenAI's GPT-4 and Google's Bard provide robust AI capabilities but rely on constant internet connectivity and come with usage costs and privacy trade-offs.

B. Offline and Local AI Models

Recent advancements in edge computing and model compression have enabled LLMs to be executed locally. Frameworks such as Ollama and GGML have made it feasible to run models like LLaMA, Mistral, and Gemma on personal computers. Research by Patel et al. (2023) emphasizes that offline AI systems enhance data privacy, lower latency, and reduce operating costs, making them ideal for decentralized and low-resource scenarios. Tools like Ollama provide RESTful APIs for real-time inference, bridging the gap between advanced AI and local applications.

C. Cross-Platform Development with Flutter

Flutter, developed by Google, has gained prominence as a cross-platform UI toolkit for building natively compiled applications from a single codebase. As per [Reis et al., 2022], Flutter's fast rendering engine and support for Windows, macOS, and Linux make it suitable for building desktop AI applications. Its built-in widgets and third-party libraries allow for the rapid creation of modern UI elements such as dynamic chat interfaces and animated transitions.

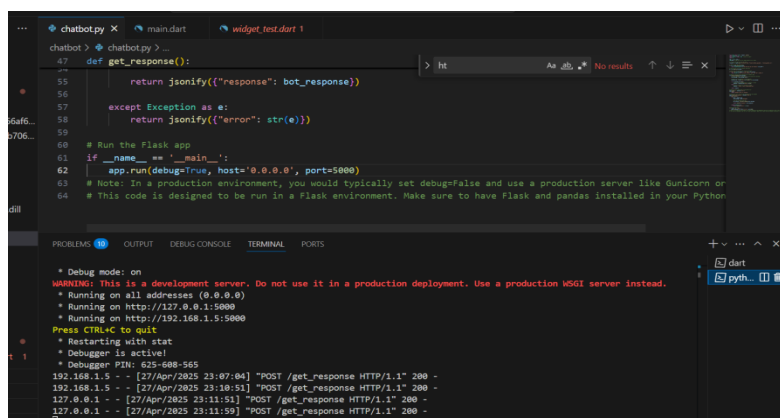
D. Chat UI/UX Trends

Modern chatbot applications like ChatGPT and Bard have set new UI/UX standards, focusing on conversational fluidity, persistent chat history, and intuitive layouts. According to Singh & Bhatia (2021), user experience plays a critical role in engagement and usability of AI assistants. Hence, adopting these design practices ensures higher user satisfaction and interaction effectiveness.

E. Gaps in Existing Research

While significant work has been done on cloud-based chatbot platforms, limited research exists on offline, local-first AI chatbot applications for desktops, particularly those utilizing open-source models and real-time user interaction. This project addresses this gap by combining local AI inference (via Ollama) with a modern UI (via Flutter), providing a secure, cost-effective, and responsive solution.

Outputs



```

def get_response():
    return jsonify({"response": bot_response})

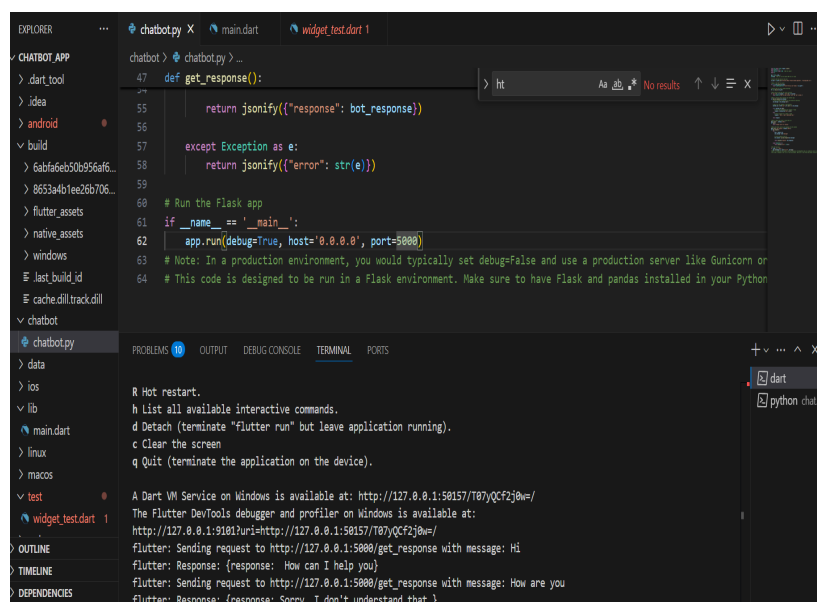
except Exception as e:
    return jsonify({"error": str(e)})

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
    # Note: In a production environment, you would typically set debug=False and use a production server like Gunicorn or
    # This code is designed to be run in a Flask environment. Make sure to have Flask and pandas installed in your Python
    
```

```

* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.5:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 625-688-565
192.168.1.5 - - [27/Apr/2025 23:07:04] "POST /get_response HTTP/1.1" 200 -
192.168.1.5 - - [27/Apr/2025 23:10:51] "POST /get_response HTTP/1.1" 200 -
127.0.0.1 - - [27/Apr/2025 23:11:53] "POST /get_response HTTP/1.1" 200 -
127.0.0.1 - - [27/Apr/2025 23:11:59] "POST /get_response HTTP/1.1" 200 -
    
```

Fig 1: Running Flask Backend Server and Testing Initial API Requests



```

def get_response():
    return jsonify({"response": bot_response})

except Exception as e:
    return jsonify({"error": str(e)})

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
    # Note: In a production environment, you would typically set debug=False and use a production server like Gunicorn or
    # This code is designed to be run in a Flask environment. Make sure to have Flask and pandas installed in your Python
    
```

```

R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen.
q Quit (terminate the application on the device).

A Dart VM Service on Windows is available at: http://127.0.0.1:50157/707yQCf2j0w=/
The Flutter DevTools debugger and profiler on Windows is available at:
http://127.0.0.1:9101?uri=http://127.0.0.1:50157/707yQCf2j0w=/
Flutter: Sending request to http://127.0.0.1:5000/get_response with message: Hi
Flutter: Response: {response: How can I help you}
Flutter: Sending request to http://127.0.0.1:5000/get_response with message: How are you
Flutter: Response: {response: Sorry, I don't understand that.}
    
```

Fig 2: Flask Server Handling Multiple Requests

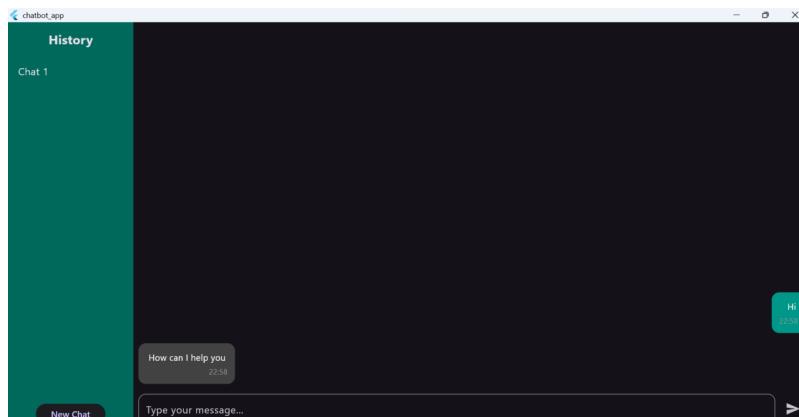


Fig 3: Chatbot Application User Interface (UI) with Chat History

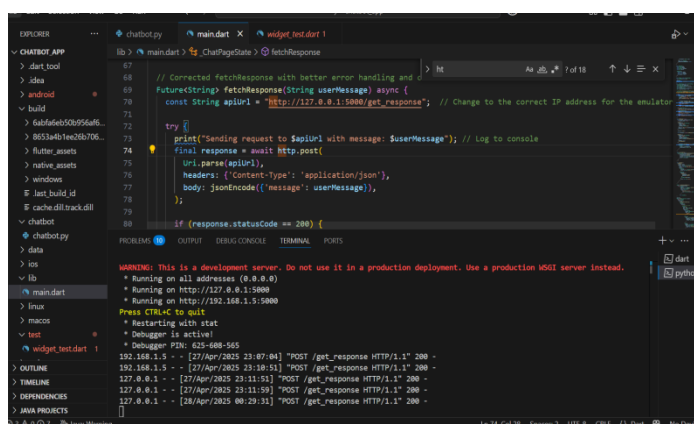


Fig 4: Flutter Frontend Code Sending API Request to Flask Server

III. CONCLUSION

In this project, a Flutter-based chatbot application was successfully developed and integrated with a locally hosted Flask server to handle user queries. The app allows users to interact through a modern, user-friendly chat interface, resembling popular AI platforms like ChatGPT and Bard. Communication between the Flutter frontend and the Flask backend was established using HTTP POST requests, ensuring real-time response handling.

Additionally, the backend was thoroughly tested using tools like Postman to validate the API responses. The project demonstrates the effective integration of a cross-platform mobile application with a Python server, providing a foundation for further enhancements such as advanced AI models, offline capabilities, and better conversation management.

This project has not only strengthened the understanding of mobile app development and backend integration but also showcased the potential for building low-cost, offline AI-based chatbot systems without relying on cloud services.

Future Enhancements :

While the current implementation of the chatbot app demonstrates the feasibility and effectiveness of using locally hosted AI models in a desktop environment, there are several opportunities for future enhancement to improve scalability, usability, and performance:

1) Multi-Model Support

Future versions of the application can allow users to switch between different locally hosted language models (e.g., LLaMA, Mistral, Gemma, Phi) based on their performance and resource availability. This flexibility would enable users to choose the most suitable model for their specific needs, whether prioritizing speed, accuracy, or memory usage.

2) Voice Input and Output Integration

To make the chatbot more accessible and interactive, voice-based communication can be incorporated. Integration with text-to-speech (TTS) and speech-to-text (STT) engines like Mozilla DeepSpeech or Whisper can facilitate hands-free conversations, which would benefit users with visual impairments or those preferring audio interaction.

3) *Persistent User Profiles and Chat Memory*

Enhancing the chatbot's ability to remember previous conversations across sessions would enable more personalized and context-aware interactions. Storing chat history and user preferences in a local database (e.g., SQLite) could simulate long-term memory, making the assistant feel more human-like and responsive.

4) *Multilingual Support*

By leveraging multilingual language models, the chatbot could support conversations in various languages, catering to a wider user base. This feature is particularly important for applications in diverse regions or educational contexts.

5) *Plugin and Tool Integration*

Inspired by modern AI tools like ChatGPT Plugins, the chatbot could be extended with plugins for features such as weather updates, file summarization, calendar integration, or email drafting—all while running locally and respecting user privacy.

6) *Mobile Platform Deployment*

Although the current version targets Windows desktops, Flutter's cross-platform capabilities allow for deployment on Android, iOS, macOS, and Linux. Expanding to mobile platforms would make the chatbot accessible to a broader audience and increase its practical utility.

7) *Model Optimization and Hardware Acceleration*

Implementing model quantization techniques and supporting GPU acceleration could significantly improve performance and reduce inference latency. This would enable smoother and faster conversations, even on low-end hardware.

8) *Local Fine-Tuning and Custom Training*

Future iterations could allow users to fine-tune the AI model with custom datasets tailored to specific domains (e.g., medical, legal, academic). This would enhance the chatbot's relevance in specialized use cases.

REFERENCES

- [1] Ms.Ch.Lavanya Susanna*1, R.Pratyusha 1, P.Swathi 2, P.Rishi Krishna 3, V.Sai Pradeep4, COLLEGE ENQUIRY CHATBOT, Volume: 07, Issue: 3 | Mar 2020.
- [2] Kooli, C. Chatbots in Education and Research: A Critical Examination of Ethical Implications and Solutions. Sustainability 2023, 15, 5614. <https://doi.org/10.3390/su15075614>
- [3] Guruswami Hiremath, Aishwarya Hajare, Priyanka Bhosale, Rasika Nanaware, Chatbot for education system, (Volume 4, Issue 3). https://www.researchgate.net/publication/347902940_Chatbot_for_Education_System
- [4] Research Paper on Chatbot Development for Educational Institute. <https://dx.doi.org/10.2139/ssrn.3861241>.
- [5] Punith S1, Chaitra B2, Veeranna Kotagi3*, Chethana R M4, Research Paper on Chatbot for Student Admission Enquiry, Volume 3 Issue 1. <https://zenodo.org/record/3733170/files/Research%20Paper%20on%20Chatbot%204-HBRP%20Publication.pdf>



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)