# A System Design for Integrating Large Language Models into Flutter Applications

Odinachi Udemezuo Nwankwo, Hee-Jae Shin, Dong-Seong Kim, Jae Min Lee
Department of IT Convergence, Kumoh National Institute of Technology, Gumi, South Korea
{odinachi, shinheejae, dskim, ljmpaul,}@kumoh.ac.kr

***Abstract*—The increasing integration of Large Language Models (LLMs) into desktop applications raises serious concerns regarding user data privacy, latency, and dependency on cloud infrastructures. This paper presents a privacy-preserving design for integrating locally hosted LLMs into Flutter applications using Ollama. The proposed architecture strictly enforces layered separation between the user interface, business logic, and data access components, ensuring that sensitive user prompts never leave the device. The design is suitable for real-world deployment in privacy-critical desktop applications.**

***Index Terms*—LLM, AI, Ollama, Flutter**

## I. Introduction

Recent advances in LLMs have enabled conversational AI systems across many application areas [1]. However, most deployments still rely on cloud-hosted models, which require sending user data to remote servers. This can create major issues such as privacy leakage [2], regulatory non-compliance, higher latency, and weak offline usability—especially for sensitive domains like healthcare, finance, and personal productivity.

Researchers in [3] show that running LLMs locally can reduce costs and broaden access to AI development, using India as their main case study. With Ollama and a study involving 180 developers, they found local deployment enabled roughly twice as many experiments and reduced costs by about 33%, while also improving learning about how AI systems work. Their mixed-method approach combined surveys, code analysis, and interviews, measuring productivity, cost, and learning outcomes. Key limitations included dependence on RTX 3060 GPUs, and some performance gaps compared to commercial APIs such as GPT-4 [3].

To address these limitations, local LLM execution has emerged as a viable alternative, enabled by efficient open-source models and lightweight inference frameworks. Nevertheless, integrating local LLMs into cross-platform mobile frameworks such as Flutter remains underexplored, especially from a software architecture and privacy-by-design perspective.

## II. Proposed Methodology

The proposed methodology adopts a layered architecture composed of the Application Layer, Business Logic Layer, Data Access Layer, and a Local LLM Server, as illustrated in Algorithm 1 and Fig. 1.

This separation enforces controlled data flow. At the Application Layer, a Flutter-based chat interface captures user

**Algorithm 1** Flutter–Ollama Design and Integration

**Define Flutter Application Layer**
    Create Chat User Interface (UI)
        *Example: Text input box and send button*
    Create State Manager
        *Example: Handles states {idle, loading, streaming, done}*
    Create Chat Controller
        *Example: Receives user message from UI*
**Define Business Logic Layer**
    Create Chat Use Case
        *Example: Rule – "Send prompt and receive AI reply"*
    Ensure UI does not communicate directly with AI server
**Define Data Access Layer**
    Create Chat Repository
        *Example: Bridges application logic and data source*
    Create Ollama Data Source
        *Example: Prepares HTTP request for local LLM*
**Define Local LLM Server (Ollama)**
    Expose API endpoint
        *Example: http://localhost:11434/api/generate*
    Load selected LLM model
        *Example: llama3*
    Enable streaming text response
**Define Integration Flow**
    UI sends user text to Controller
        *Example: "Hello"*
    Controller calls Chat Use Case
    Use Case requests response from Repository
    Repository forwards request to Ollama Data Source
    Data Source sends HTTP request to Ollama API
    Ollama streams generated text back
        *Example: "Hi", then "Hi there"*
    Response flows back through Repository and Use Case
    State Manager updates UI with streaming text
**Enforce Design Constraints**
    All AI processing runs locally
    No user data is sent to cloud servers
    Components communicate only through defined layers
**End System Design**

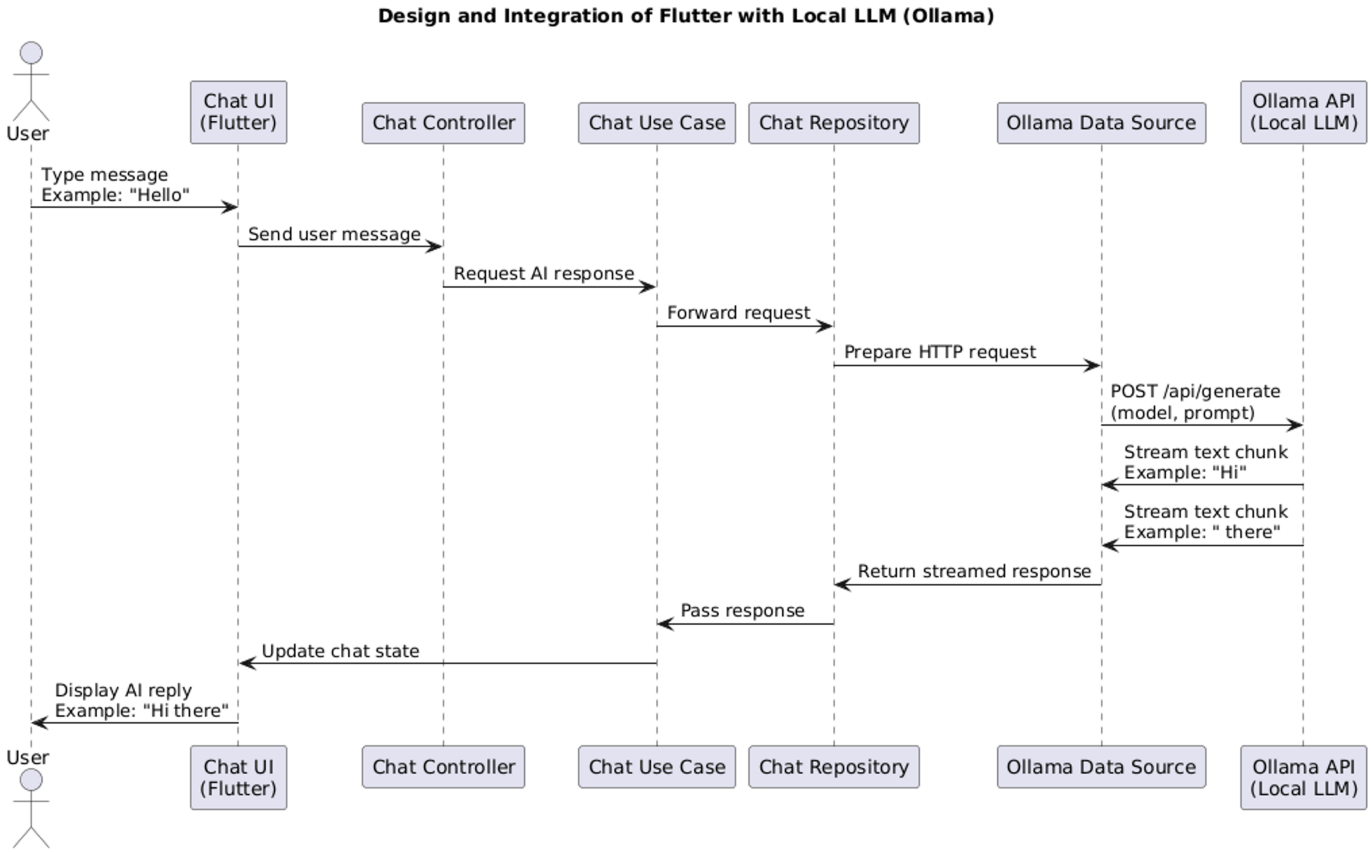


Fig. 1. Sequence Diagram of the Design

input and displays streaming responses. State management governs interaction states such as idle, loading, streaming, and completion, ensuring responsive user experience. The UI communicates exclusively with a Chat Controller, eliminating any direct interaction with the AI backend.

The Business Logic Layer encapsulates domain rules through a Chat Use Case, which defines the prompt-response workflow. This abstraction ensures that application logic remains independent of the underlying AI implementation, improving modularity and testability.

The Data Access Layer serves as a secure bridge between application logic and the AI engine. A Chat Repository forwards requests to an Ollama [4] [5] Data Source, which constructs HTTP requests targeting a local Ollama API endpoint. The Local LLM Server hosts the selected model (e.g., LLaMA-based variants) and streams generated tokens incrementally back to the application.

## III. Conclusion and Future Work

This paper presented a privacy-preserving design steps for integrating local Large Language Models into Flutter applications using Ollama. By enforcing layered separation and executing all AI processing on-device, the proposed system eliminates cloud dependency, enhances user privacy, and enables offline intelligent interactions. Future work will focus on implementing the design steps proposed here to achieve the first prototype.

## IV. Acknowledgement

This work was partly supported by Innovative Human Resource Development for Local Intellectualization program through the IITP grant funded by the Korea government(MSIT) (IITP-2025-RS-2020-II201612, 33%) and by Priority Research Centers Program through the NRF funded by the MEST(2018R1A6A1A03024003, 33%) and by the MSIT, Korea, under the ITRC support program(IITP-2025-RS-2024-00438430, 34%).

## References

[1] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, and J. Chen, "A review on edge large language models: Design, execution, and applications," *ACM Computing Surveys*, vol. 57, no. 8, pp. 1–35, 2025.

[2] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, and X. Cheng, "On protecting the data privacy of large language models (llms) and llm agents: A literature review," *High-Confidence Computing*, vol. 5, no. 2, p. 100300, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667295225000042

[3] V. Udandarao and N. Misra, "Democratizing ai development: Local llm deployment for india's developer ecosystem in the era of tokenized apis," 2025. [Online]. Available: https://arxiv.org/abs/2508.16684

[4] J. Gohil, H. L.Shifare, and M. Shukla, "Developing a user-friendly conversational ai assistant for university using ollama and llama3," in *2025 International Conference on Data Science, Agents Artificial Intelligence (ICDSAAI)*, 2025, pp. 1–5.

[5] M. A. J, M. A. VR, M. K, and M. P. R, "Redact: Pii redaction and privacy protection with ollama integration," in *2025 8th International Conference on Computing Methodologies and Communication (ICCMC)*, 2025, pp. 577–583.