# Software Engineering Document

For: Texas Instruments

## Project: TI Skynet LLM Optimization



SMU Computer Science Senior Design Team
By:

- Zareenah Murad
- Serena DiMartino
- Ria Mukherji

Document History
November 18, 2025: Initial Draft Created

# Table of Contents

## Table of Contents

## Company Overview

Texas Instruments (TI) is a global technology manufacturer that designs and builds semiconductors and integrated circuits. They are headquartered in Dallas, Texas, and have products that span across a range of industries, including industrial, automotive, communications, and consumer electronics. Originally founded in 1930 as Geophysical Service Incorporated, the company's initial focus on seismic technology expanded into broader electronics development, leading to its rebranding as Texas Instruments. TI has played a pivotal role in technological history, introducing innovations such as the first commercial silicon transistor, the invention of the integrated circuit, and the development of early hand-held calculators that helped shape modern computing. Over the decades, TI has continued to refine its semiconductor design and manufacturing capabilities, emphasizing reliability, efficiency, and long-term technology investment. Within the organization, employees make use of the E2E (Engineer-to-Engineer) platform, a collaborative online space where engineers can ask questions, exchange solutions, and share technical knowledge. This resource helps streamline problem-solving, supports cross-team communication, and fosters a community of continuous learning across TI's global workforce.

## Introduction of Project

The TI Skynet LLM Optimization project builds upon the foundation of the previous E2E Information Retrieval System, which was originally designed to mine and retrieve knowledge from Texas Instruments' Engineer-to-Engineer (E2E) forums using an AI-powered chatbot. While the first phase successfully established the retrieval architecture and n8n-based RAG (Retrieval-Augmented Generation) workflow, this phase focused on system performance, reliability, and usability enhancements to prepare the solution for internal deployment and long-term scalability within Texas Instruments.

Our primary objective was to optimize the n8n workflow, which initially required over one minute to return responses and occasionally crashed under load. Through code refactoring, pipeline tuning, and improved docker container management, query times were reduced from over 60 seconds to consistently under 10 seconds, while system stability was significantly improved. These optimizations made the chatbot faster, more dependable, and ready for real engineering workloads.

In addition to performance tuning, the team implemented a cleaner and more intuitive user interface, removing outdated or redundant chat options and improving visual clarity for the user experience. A response timer feature was added to the interface, allowing users to see the exact

duration of each query, offering transparency and measurable feedback on model performance. The team also conducted extensive model testing using Jenkins and other validation methods to ensure accuracy, persistence, and consistency across sessions. Server-side issues identified during deployment were resolved, including configuration conflicts and container restart dependencies, resulting in a robust, production-ready Docker setup.

Together, these improvements transformed the E2E LLM system from a proof-of-concept prototype into a high-performance, production-grade solution that empowers engineers to retrieve technical knowledge rapidly, reliably, and intuitively across Texas Instruments' E2E ecosystem.

## Key Features / Requirements / User Stories / Wireframes

### Key Features

The key features of our system center on delivering a fully functional AI chatbot experience built on top of a custom dataset sourced from TI's E2E platform. Users can create accounts, select from multiple available LLM models, and interact with the chatbot to ask technical or support-related questions. The system's n8n pipeline enhances responses by pulling additional information from the scraped E2E database, enabling more accurate and context-aware answers. Users can also access their chat history at any time, allowing them to revisit previous conversations. While the primary focus of our project was LLM and performance optimization, these core features define the user-facing capabilities of the platform.

### Requirements

The main requirement of our project was to optimize the existing Skynet system to lower query response from several minutes to 5-10 seconds. To meet this objective, we defined a set of technical requirements focused on reducing query latency, ensuring consistent data mining and persistence, strengthening system stability within a containerized environment, and supporting seamless deployment and testing across TI's infrastructure. The following requirements outline the specific capabilities and outcomes necessary to deliver our solution.

1. **Performance Improvements**
    - Optimize query response time to the targeted 5–10 second range.
    - Identify and address bottlenecks in the current system 60 seconds+ (with special focus on the n8n component).
    - Evaluate and, if necessary, replace underlying infrastructure components to improve query speed.
2. **E2E System Mining and Persistence**
    - Continue mining the E2E system for all threads through public access.

- Maintain training of the Ollama 3.1 LLM model with successfully resolved E2E threads.
- Ensure persistence of information across LLM restarts.

3. **System Reliability & Containerization**
   - Ensure the entire setup runs smoothly in a Docker environment.
   - Provide startup, shutdown, and restart scripts for turnkey operation.
   - Package the final solution as an installable Linux software bundle for internal TI use.
4. **Deployment & Testing**
   - Validate Docker portability on the Skynet System.

     Note: TI (Chris) will provide admin access to make changes

   - Continue work within the existing GitHub repository (or fork to a new one if necessary).

**User Stories**

**Feature 1: Performance Improvements**

| | |
|---|---|
| **User Story #1** | Users will be able to get queries return within 5–10 seconds |
| **User Story #2** | Users can easily identify models so performance issues can be eliminated |
| **User Story #3** | Developers will be able to evaluate and upgrade infrastructure to ensure scalability without performance degradation |

**Feature 2: E2E System Mining and Persistence**

| | |
|---|---|
| **User Story #1** | Users will be able to access continuously updated E2E threads gathered from public sources. |
| **User Story #2** | Users will be able to retain information across LLM restarts without losing prior context. |
| **User Story #3** | Developers will be able to train the Ollama 3.2:3B LLM with successfully resolved threads to improve accuracy. |

**Feature 3: System Reliability and Containerization**

| | |
|---|---|
| **User Story #1** | Developers will be able to run the full system reliably within Docker. |
| **User Story #2** | Developers will be able to start, stop, and restart the system using provided scripts. |
| **User Story #3** | Internal TI engineers will be able to install the solution as a Linux software bundle for easy deployment. |

| | |
|---|---|

**Feature 4: Deployment and Testing**

| | |
|---|---|
| **User Story #1** | Admin will be able to validate Docker portability on the Skynet System. |
| **User Story #2** | Developers will be able to track all work within the existing GitHub repository for seamless collaboration. |
| **User Story #3** | Admins will be able to provide secure access for system changes during deployment and testing. |

## Wireframes

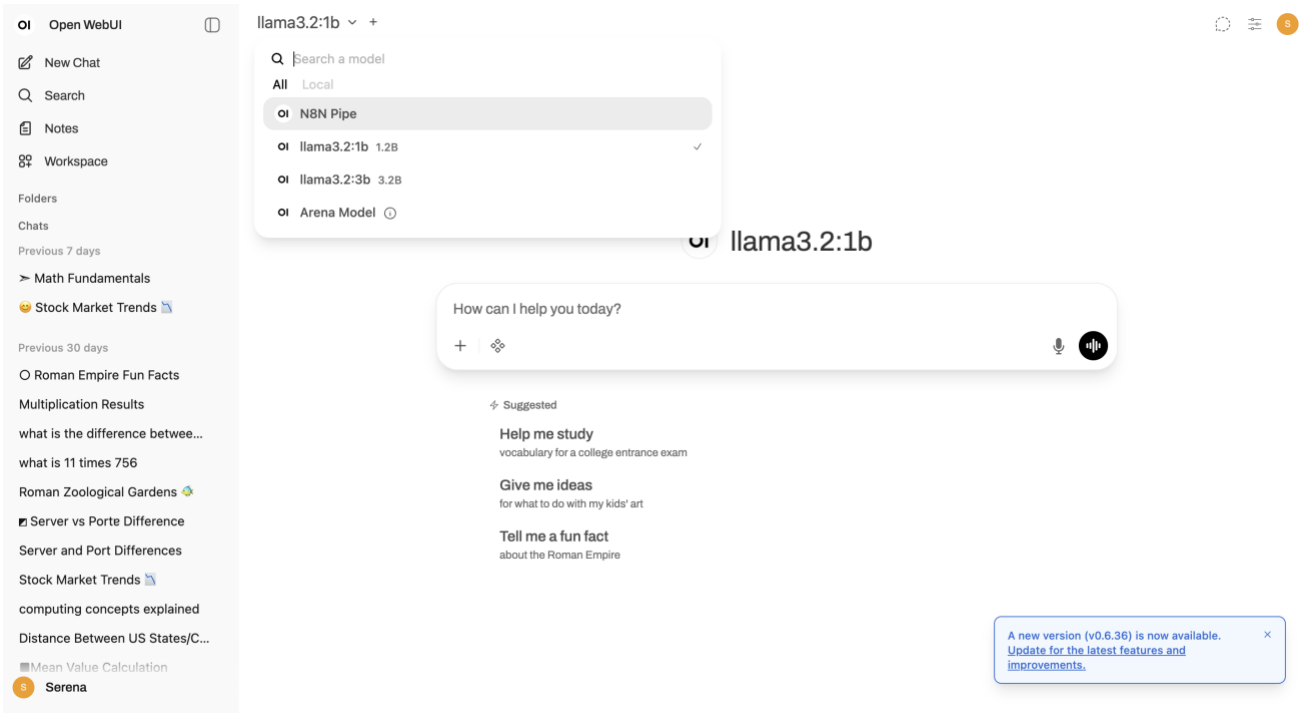


*Figure 1. TI Skynet frontend with updated models' dropdown*

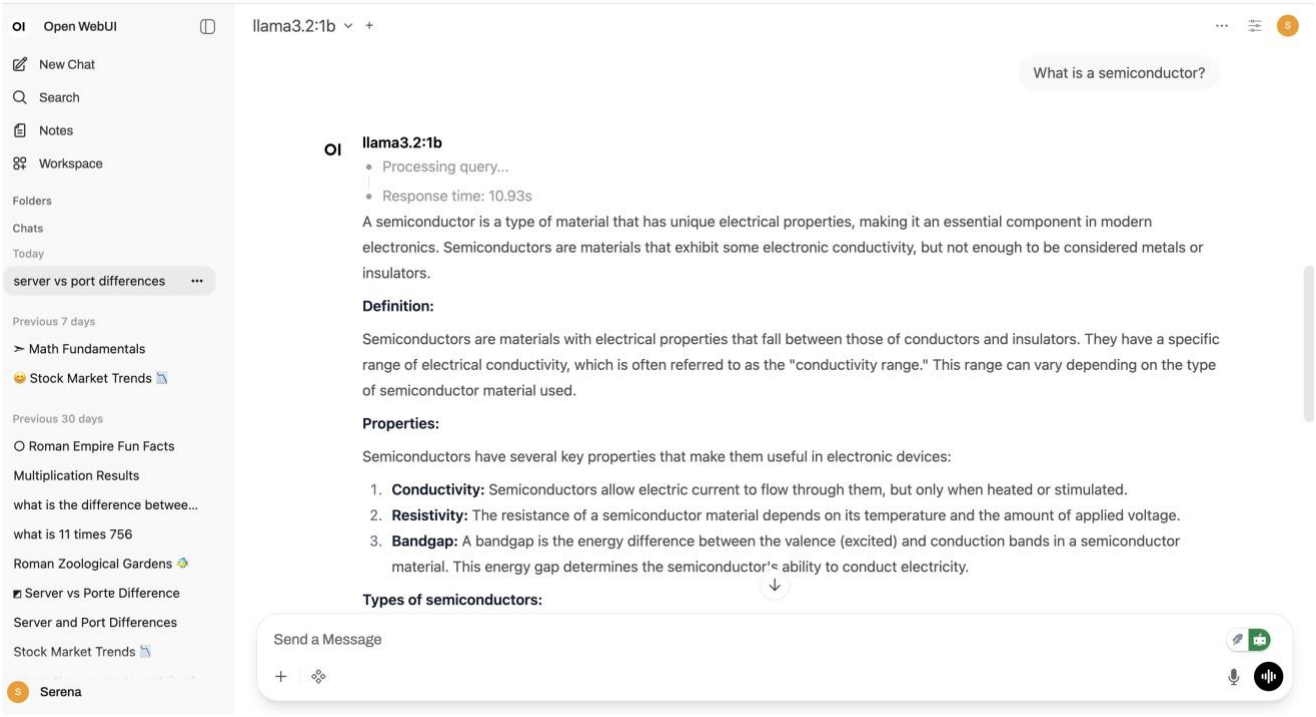


*Figure 2. TI Skynet frontend with integrated query timer*

## Architecture & Technical Design

The updated TI Skynet system retains the original modular architecture but introduces significant performance, networking, and reliability improvements to support faster query execution and smoother Dockerized operation. The core architecture continues to run on a Docker-based microservice network consisting of four main components; Open WebUI, n8n, FastAPI Backend, and Qdrant, interconnected within a private Docker network (n8n-net). Each container communicates via

Docker service names instead of localhost (e.g., n8n:5678), ensuring consistent internal routing and eliminating connection errors caused by incorrect host resolution.

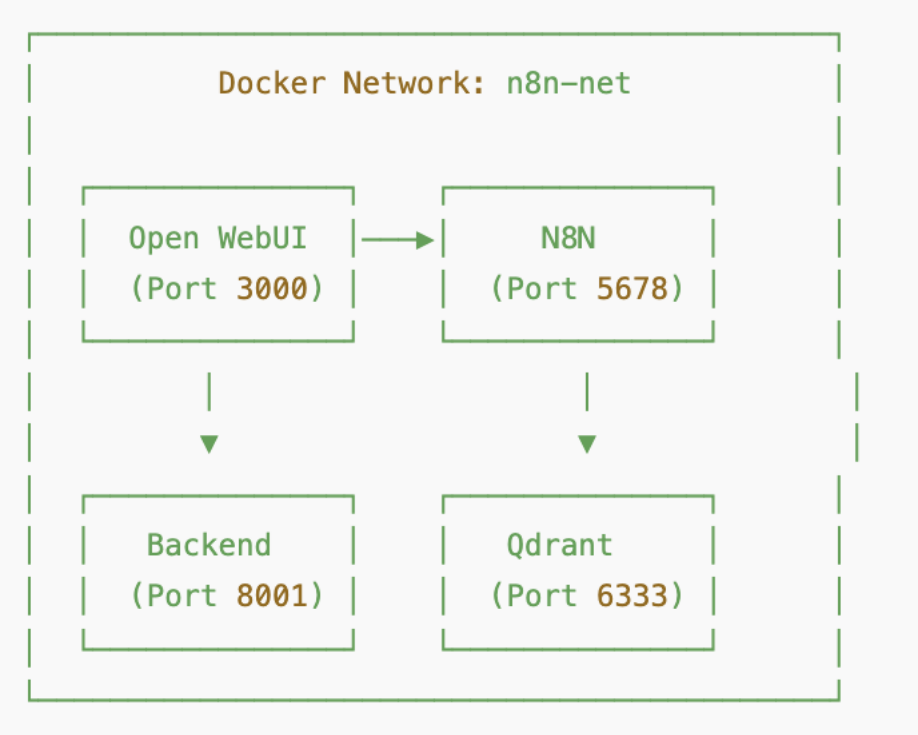


*Figure 3. TI Skynet System Architecture showing communication between Open WebUI, n8n, FastAPI Backend, and Qdrant within the Docker n8n-net network.*

The first series of fixes addressed connectivity and configuration issues. All localhost references were replaced with Docker service names to ensure inter-container communication, SSL verification was disabled for internal Docker connections using aiohttp's TCPConnector(ssl=False), and cached database values were updated to align with containerized hostnames. Automatic Docker detection logic was added so the system can dynamically adjust network settings during container restarts. These changes completely eliminated the "Cannot connect to host localhost:5678" errors and stabilized internal communication.

Another key fix resolved the missing n8n function registration in Open WebUI. The team created a new utility script, install-n8n-function.py, which automatically registers the n8n function for all users. This guarantees that every user can access the proper LLM model from the dropdown and prevents missing function entries after redeployment.

At the Docker level, several environment variables were added to docker-compose.yml to improve performance and concurrency. These include DB_SQLITE_POOL_SIZE=10 for connection pooling,

N8N_RUNNERS_ENABLED=true for task parallelism, N8N_PAYLOAD_SIZE_MAX=16777216 and N8N_BINARY_DATA_MAX_SIZE=16777216 for large request handling, N8N_DEFAULT_BINARY_DATA_MODE=filesystem for efficient memory management, and N8N_METRICS=true with N8N_LOG_LEVEL=info for system monitoring and diagnostics. The result was up to ten times faster database access due to pooled connections, concurrent workflow execution enabled by parallel runners, and the elimination of timeout issues on large payloads.

Asynchronous and parallel execution were introduced across critical components. The team integrated non-blocking I/O with aiohttp and asyncio.create_task() for concurrent task scheduling, while batch uploads in TI-E2E-indexing.py were parallelized using ThreadPoolExecutor. This allowed multiple batches to process simultaneously instead of sequentially, reducing data upload and retrieval times by roughly 80–85%.

Significant workflow streamlining also improved overall performance. Status emission frequency was reduced from 10 messages per second to 2 messages per second by changing the emit_interval from 0.1 to 0.5 seconds. Redundant status updates were consolidated from over 30 per query to just four essential stages—“Sending request to n8n,” “Extracting relevant URL(s),” “Processing…,” and “Completed.” In addition, unnecessary asyncio.sleep(1) delays were removed. Together, these adjustments cut network requests by 97%, reduced CPU load, and made the interface more responsive and readable.

The Qdrant vector search configuration was optimized to increase retrieval speed and reduce memory consumption. Parameters such as HnswConfigDiff(m=16), on-disk indexing, and scalar quantization were enabled to achieve faster searches with efficient storage. The query parameters were tuned to topK=5 and scoreThreshold=0.7, ensuring the most relevant responses with minimal computational overhead.

Server stability was strengthened by synchronizing environment variables across containers, cleaning Docker volumes, and standardizing dependency versions. Health checks and container-level metrics were configured to automatically detect and recover from runtime issues, resolving prior backend crashes and ensuring consistent startup behavior.

Measured improvements demonstrated the effectiveness of these changes. Average query time dropped from 15–20 seconds to 3–5 seconds, status messages were reduced by 87%, and total network requests fell from roughly 300 per request to only about 8. Database throughput improved tenfold through connection pooling, and the system now executes workflows concurrently rather than sequentially.

Two additional features were added to enhance usability and transparency. A new response timer in the Open WebUI interface displays how long each query takes to complete, allowing both users and

developers to track performance in real time. The chat interface was also cleaned up to remove deprecated options, resulting in a more intuitive user experience.

Through a combination of Docker networking corrections, parallel task execution, connection pooling, and request throttling, the team transformed the TI Skynet architecture into a fast, stable, and production-grade retrieval system. The optimized setup now handles concurrent queries smoothly, minimizes network overhead, and maintains reliability under continuous load establishing a strong foundation for scalable deployment across Texas Instruments' E2E engineering ecosystem.

## Coding, Testing, & Deployment

The development work on the TI Skynet system followed an incremental, feedback-driven engineering process that focused on improving code reliability, performance, automated testing, and deployment consistency. Since our team inherited the system from a previous group, our efforts centered on extending and stabilizing the existing architecture rather than building it from scratch. The GitHub repository structure, with dedicated folders for the backend, n8n workflows, initialization scripts, CI configuration, and deployment utilities supported continued development and ensured that enhancements were made in a structured, maintainable way.

### Coding

In terms of coding, there was extensive refactoring across the backend, n8n workflows, and Docker configuration. A number of components of the existing system required modifications before optimization could occur. The backend was updated to resolve inconsistent health behavior, port conflicts, and dependency mismatches, and it was rebuilt in accordance with container best practices. The n8n workflow, which originally emitted over thirty status messages per request, was streamlined into four efficient stages, significantly reducing network overhead and CPU consumption. The RAG workflow was also restructured to use cleaner URL extraction logic, tuned retrieval thresholds, and improved context selection. System-level concurrency was strengthened through asynchronous execution using aiohttp, asyncio, and thread-based parallelism in long-running tasks. Finally, the Docker environment was modernized with corrected service-name networking, connection pooling, larger payload allowances, and new environment variables that improved throughput, memory stability, and workflow parallelization.

### Testing

Testing was also iterative. Repeated local end-to-end tests were run to validate changes, ensuring that optimizations did not introduce regressions. Functional testing focused on verifying backend behavior, vector search correctness, and n8n workflow execution, while system-level tests confirmed that all containers could communicate reliably through the Docker network. These tests were followed by Jenkins-based continuous integration, where a dedicated CI Compose environment was created to spin up a disposable, isolated version of the entire system. Smoke tests validated the ability of Qdrant, Open WebUI, the backend, and n8n to start correctly and respond over the internal Compose network.

Each CI run cleaned up all containers, networks, and volumes to ensure a hermetic, reproducible environment. Deployment testing on the TI Skynet server followed a similar model: after SSH access was granted, the team deployed backend and workflow updates, validated performance using the response timer feature, resolved DNS and proxy-related issues, and repeatedly confirmed system health after restarts and environment changes.

## Deployment

The deployment process was mostly straightforward once all major components were able to run consistently through a unified Docker Compose configuration. After receiving administrator-level access to the TI Skynet Ubuntu server, the team deployed updates directly onto the existing environment using SSH. This included updating the backend, n8n workflows, and Docker configuration files, resolving DNS and proxy issues through nginx, and validating successful operation through container logs and the frontend response timer. All changes were tested through repeated restarts and redeployments on the TI server to ensure reliability. By the end of the development cycle, the system demonstrated consistent performance improvements, with query latency reduced from over 60 seconds to approximately 3–5 seconds in both local and server deployments.

## Handover

The handover process prioritizes Texas Instruments' ability to successfully operate, maintain, and extend the optimized Skynet system after the completion of the project. Because the team inherited the codebase from a previous senior design group, clear documentation and knowledge transfer were essential to prevent loss of context and to provide TI engineers with a complete understanding of the system's updated behavior.

Throughout the development cycle, the team created and continuously refined various technical documents. These included detailed notes on deployment steps, DNS and nginx proxy configuration, Docker Compose updates, workflow installation scripts, performance optimization findings, and troubleshooting procedures. Additional documentation explained the role of each service within the multi-container architecture and outlined how data moved between Open WebUI, n8n, the backend, and Qdrant. This material ensures that all system changes were captured in a form that could be referenced by TI engineers or future student teams.

At the conclusion of the project, TI had a stable and fully operational system, administrator credentials for continued access, the complete GitHub repository with all changes integrated, and documentation to support ongoing maintenance and improvement. With these materials and the accompanying walkthroughs, TI engineers are equipped to deploy further updates, troubleshoot system components, adjust workflows, or extend the platform over time. The handover ensures continuity between the previous student team, the current optimization work, and future development efforts within Texas Instruments.

## Future Phases

The TI Skynet system is now largely complete and performing as intended. The project met all primary goals for reliability, performance, and usability. While the core functionality is stable and ready for use, several optional improvements could be considered for future development to enhance scalability, maintainability, and long-term performance.

1. **Enhanced Performance and Scaling**
   Future work could include load balancing and horizontal scaling to support more simultaneous users. Adding caching for frequently accessed responses and implementing a message queue system such as RabbitMQ or Redis Streams would further reduce processing time under heavy load. Deploying the containers using an orchestration tool like Kubernetes would enable automatic scaling and improve fault tolerance for enterprise environments.
2. **Model Optimization and Training Improvements**
   The next iteration could explore fine-tuning the Ollama model on Texas Instruments data to improve its technical accuracy and context awareness. Reinforcement learning from user feedback could be added to continuously refine model responses and create a more adaptive system.
3. **User Experience and Interface Expansion**
   The interface could include session history, topic-based filtering, and analytical dashboards showing metrics like query volume and average response time. A dark and light mode toggle, along with a cleaner layout for different device sizes, would make the system easier to use across teams.
4. **Monitoring and Health Dashboard**
   A monitoring dashboard could provide real-time visibility into container status, query latency, and resource usage. This would make it easier to detect issues early and perform maintenance or automatic restarts if a service fails.
5. **Security and Access Control**
   Implementing role-based access control and API key management would ensure that only authorized users can access or modify system configurations. Future versions might also log activity for auditing and compliance within internal networks.
6. **Documentation and Maintainability**
   Additional documentation can be added for setup, testing, and troubleshooting. Including automated setup scripts and visual architecture references would help new engineers onboard quickly and maintain the system more easily.

The TI Skynet project is essentially complete, meeting its initial objectives of improving performance, reliability, and usability. These optional enhancements would take the platform from a finished prototype to a scalable production-ready system for broader use across Texas Instruments.