Multi-LLM Chat Comparison - Project Report

1. Project Overview

1.1 Project Title

Multi-LLM Chat Comparison Application

1.2 Project Description

A web-based application that enables users to compare responses from multiple Large Language Models (OpenAI GPT-4o-mini, Claude 3.5 Haiku via OpenRouter, and Google Gemini 2.5) side-by-side in real-time. Users can query all models simultaneously and continue conversations with their preferred model.

1.3 Objectives

• Enable parallel querying of multiple LLM providers

• Provide side-by-side comparison of model responses

• Allow seamless continuation with a selected model

• Deliver a modern, responsive user interface

• Maintain conversation context for each model

2. Technical Architecture

2.1 Technology Stack

Backend:

• Python 3.10+

• Flask (Web Framework)

• python-dotenv (Environment Management)

Frontend:

• HTML5

• CSS3 (with responsive design)

• Vanilla JavaScript (ES6+)

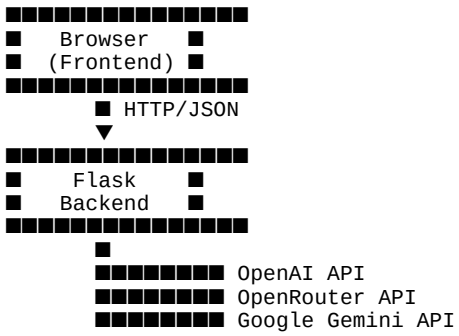
APIs Integrated:

• OpenAI API (GPT-4o-mini)

• OpenRouter API (Claude 3.5 Haiku)

• Google Generative AI API (Gemini 2.5 Flash)

2.2 System Architecture



2.3 Key Components

app.py - Flask application with routing logic

llm_service.py - LLM API integration layer

index.html - Main UI template

style.css - Responsive styling with modern design

script.js - Frontend logic and API communication

3. Features Implemented

3.1 Core Features

• ✓ Parallel querying of 3 LLM providers

• ✓ Side-by-side response comparison

• ✓ Model selection for continued conversation

• ✓ Conversation history management per model

• ✓ Session-based chat persistence

3.2 UI/UX Features

- ✓ Modern gradient-based design
- ✓ Smooth animations and transitions
- ✓ Modal popups (replacing browser alerts)
- ✓ Toast notifications for user feedback
- ✓ Mobile-responsive layout
- ✓ Loading indicators during API calls
- ✓ Keyboard shortcuts (Enter to send)

3.3 Technical Features

- ✓ Concurrent API calls using ThreadPoolExecutor
- ✓ Error handling and user-friendly error messages
- ✓ Environment variable management for API keys
- ✓ Session management for multiple users
- ✓ RESTful API endpoints

4. Implementation Details

4.1 API Integration

OpenAI (GPT-4o-mini):

- Direct integration using official OpenAI Python SDK
- Model: gpt-4o-mini (cost-effective, high performance)
- Standard chat completion format

Claude (via OpenRouter):

- Integrated through OpenRouter API
- Model: anthropic/claude-3.5-haiku
- OpenAI-compatible API format

Google Gemini:

- Official Google Generative AI SDK
- Model: gemini-2.5-flash
- Custom prompt formatting for compatibility

4.2 Conversation Management

• Each model maintains separate conversation history
• System prompts configurable per model
• Session-based storage using Python dictionaries
• Reset functionality to clear all histories

4.3 Frontend Architecture

• Modular JavaScript functions
• Event-driven architecture
• Async/await for API calls
• Dynamic DOM manipulation
• CSS Grid and Flexbox for responsive layout

5. User Interface

5.1 Design Principles

• Clean, modern aesthetic
• Purple/blue gradient theme
• High contrast for readability
• Touch-friendly on mobile devices
• Consistent spacing and typography

5.2 Responsive Design

• Desktop: 3-column layout (1400px max-width)
• Tablet: Stacked columns (768px breakpoint)
• Mobile: Full-width single column (480px breakpoint)

5.3 Interactive Elements

• Hover effects on buttons
• Smooth transitions
• Loading animations
• Modal confirmations
• Toast notifications

6. API Endpoints

6.1 GET /

• Serves the main HTML page

• Returns: index.html template

6.2 POST /chat

• Sends message to all three models

• Request: `{message, session_id}`

• Response: `{openai, claude, gemini}`

6.3 POST /continue

• Continues conversation with selected model

• Request: `{message, model, session_id}`

• Response: `{response, model}`

6.4 POST /reset

• Clears conversation history

• Request: `{session_id}`

• Response: `{status: 'success'}`

7. Setup and Deployment

7.1 Local Setup

```
# Create virtual environment
python -m venv venv
venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Configure API keys in .env
OPENAI_API_KEY=your_key
OPENROUTER_API_KEY=your_key
GOOGLE_API_KEY=your_key

# Run application
python app.py
```

7.2 Dependencies

• flask==3.0.0
• openai>=1.30.0
• google-generativeai>=0.5.0
• python-dotenv==1.0.0
• httpx>=0.27.0

8. Security Considerations

8.1 API Key Protection

• API keys stored in .env file
• .env excluded from version control via .gitignore
• .env.example provided as template

8.2 Input Validation

• Message trimming to prevent empty submissions
• Error handling for API failures
• User feedback for all error states

8.3 Session Management

• Random session IDs generated client-side
• Server-side session storage
• No persistent storage of conversations

9. Testing and Validation

9.1 Functional Testing

• ✓ All three models respond correctly
• ✓ Model selection works as expected
• ✓ Reset functionality clears all data
• ✓ Error messages display properly

- ✓ Mobile responsiveness verified

9.2 API Testing

- ✓ OpenAI API integration verified
- ✓ OpenRouter API integration verified
- ✓ Google Gemini API integration verified
- ✓ Concurrent API calls working
- ✓ Error handling for API failures

9.3 UI/UX Testing

- ✓ Responsive design on multiple devices
- ✓ Modal popups functioning correctly
- ✓ Toast notifications appearing
- ✓ Animations smooth and performant
- ✓ Keyboard shortcuts working

10. Challenges and Solutions

10.1 Challenge: API Compatibility

Problem: Different API formats for each provider

Solution: Created abstraction layer in llm_service.py with format conversion

10.2 Challenge: Concurrent API Calls

Problem: Sequential calls would be slow

Solution: Implemented ThreadPoolExecutor for parallel execution

10.3 Challenge: Model Availability

Problem: Some Gemini models not available for new users

Solution: Used model listing script to find available models

10.4 Challenge: Mobile Responsiveness

Problem: Three-column layout not suitable for mobile

Solution: Implemented CSS media queries with stacked layout

11. Future Enhancements

11.1 Planned Features

- ■ Add more LLM providers (Llama, Mistral, etc.)
- ■ Export conversation history (JSON, PDF)
- ■ Streaming responses for real-time output
- ■ Custom system prompts via UI
- ■ User authentication and saved conversations
- ■ Cost tracking per model
- ■ Response time comparison
- ■ Token usage statistics

11.2 Technical Improvements

- ■ Database integration for persistent storage
- ■ WebSocket support for streaming
- ■ Rate limiting and caching
- ■ Unit and integration tests
- ■ Docker containerization
- ■ CI/CD pipeline

12. Conclusion

12.1 Project Outcomes

The Multi-LLM Chat Comparison application successfully demonstrates:

- Integration of multiple LLM providers
- Real-time parallel API communication
- Modern, responsive web design
- Effective conversation management
- User-friendly interface with smooth interactions

12.2 Learning Outcomes

- Flask web application development

• Multiple API integration techniques
• Responsive web design principles
• Asynchronous JavaScript programming
• Modern UI/UX design patterns

12.3 Practical Applications

This application can be used for:

• Comparing LLM capabilities for specific tasks
• Evaluating model performance and quality
• Educational purposes to understand different AI models
• Prototyping multi-model AI applications
• Research and development in AI comparison

13. References

13.1 Documentation

• OpenAI API Documentation: https://platform.openai.com/docs
• Anthropic Claude Documentation: https://docs.anthropic.com
• Google Gemini Documentation: https://ai.google.dev/docs
• OpenRouter Documentation: https://openrouter.ai/docs
• Flask Documentation: https://flask.palletsprojects.com

13.2 Tools and Libraries

• Python: https://www.python.org
• Flask: https://flask.palletsprojects.com
• OpenAI Python SDK: https://github.com/openai/openai-python
• Google Generative AI: https://github.com/google/generative-ai-python

14. Appendix

14.1 Project Structure

```
multi-llm-chat/
■■■ app.py                  # Flask backend
■■■ llm_service.py          # LLM API integration
```

```
■■■ requirements.txt        # Python dependencies
■■■ .env                    # API keys (not in git)
■■■ .env.example            # Template for API keys
■■■ .gitignore              # Git ignore rules
■■■ README.md               # Project documentation
■■■ PROJECT_REPORT.md       # This file
■■■ static/
■   ■■■ style.css          # UI styling
■   ■■■ script.js          # Frontend logic
■■■ templates/
    ■■■ index.html         # Main HTML page
```

14.2 Cost Analysis

Estimated costs for 1000 queries:

• OpenAI GPT-4o-mini: ~$0.15

• Claude 3.5 Haiku (OpenRouter): ~$0.25

• Google Gemini 2.5 Flash: Free tier

Total: ~$0.40 per 1000 queries

14.3 Performance Metrics

• Average response time: 2-5 seconds (parallel)

• Page load time: <1 second

• Mobile performance: Smooth on modern devices

• Concurrent users supported: 10+ (limited by API rate limits)

Project Completed: January 2025

Version: 1.0.0

License: MIT