ULFG III

TASKOPTIMA

Mini-project

by

Mohammad Kansoun, 6123

Spring 2024-2025

Supervisor:

Prof Mohamad AOUDE

1. Introduction

TaskOptima is a full- stack task management system augmented by a local Large Language Model (LLM) to provide intelligent task prioritization, breakdowns, and scheduling suggestions. Developed as an end- to- end proof- of- concept, the system demonstrates how open- source AI and modern web frameworks can deliver a robust productivity assistant without relying on externally hosted model APIs.

2. Project Background & Objectives

- **Motivation**: Many to- do apps lack true intelligence. We sought to embed AI-driven decision support directly alongside CRUD operations.
- **Goals**:
    1. Provide seamless task creation, completion, and cleanup via REST endpoints.
    2. Integrate a local, quantized LLaMA model for next- task recommendations, breakdowns, and scheduling.
    3. Maintain conversation history for transparency and context.
    4. Package entire stack as Docker containers for reproducibility.

3. Technical Stack

Layer	Technology	Role
Backend API	FastAPI + Uvicorn	HTTP endpoints, dependency injection
ORM & Database	SQLAlchemy + psycopg2	PostgreSQL interaction
Environment	python-dotenv	Configuration via `.env`
AI Inference	llama-cpp-python, LangChain	Local Phi- 2 LLM integration
Frontend UI	React.js + Axios + Bootstrap	Interactive task UI and AI chat
Containers	Docker + Docker Compose	Service isolation and orchestration
LLM Model Format	GGUF Q4_K_M	Quantized model for CPU inference

4. Architecture & Workflow & Project Flow

1. **User Interaction (React Frontend)**
    - User adds, views, completes, or removes tasks.
    - User asks AI by typing questions or clicking buttons (e.g., “Suggest Weekly Plan”).

2. **API Calls & Routing (FastAPI)**
    - React uses Axios to send HTTP requests to:
        - `/tasks` endpoints for CRUD operations.
        - `/agent/query` for general AI advice.
        - `/agent/schedule` for weekly planning.
        - `/agent/history` for conversation logs.
3. **Database Operations (SQLAlchemy + PostgreSQL)**
    - Tasks are stored in a `tasks` table with fields: title, priority, deadline, completed, created_at.
    - AI Q&A pairs are logged in a `conversations` table for history retrieval.
4. **Prompt Construction (LangChain Utils)**
    - `get_urgent_overdue_tasks` flags overdue and high-priority items.
    - `taskoptima_prompt` assembles the task list, alerts, and instructions into a single prompt.
    - Dedicated prompts for scheduling (`/agent/schedule`) bypass the general prompt logic.
5. **LLM Inference (llama-cpp-python + Phi-2 Model)**
    - Prompts are sent to the quantized Phi-2 LLaMA model via `ask_phi2()`.
    - Responses are returned as JSON to the frontend.
6. **Response & UI Update**
    - AI responses displayed in the chat section.
    - Weekly plans and breakdowns rendered below controls.
    - Conversation history refreshed automatically after each AI query.
7. **Containerization & Deployment**
    - Docker Compose brings up three services: `db`, `backend`, and `frontend`.
    - Ensures environment consistency and simplifies deployment.

5. Dual AI Implementation Approaches

To maximize flexibility, two AI invocation strategies were prototyped: - **Local LLM (Primary)**: Runs quantized Phi- 2 model on CPU via `llama-cpp-python`. Ensures data privacy and offline capability.
- **Cloud API (Alternative)**: Integration points defined for switching to hosted LLM

endpoints (e.g., OpenAI GPT) via LangChain's `OpenAI()` wrapper if GPU or higher- capacity model needed.

6. Ethical Considerations

- **Data Privacy**: All prompts and responses remain local by default; no user data is sent externally.
- **Transparency**: Conversation logs allow users to review AI advice history.
- **Bias & Limitations**: Quantized LLMs can reflect training biases; users are warned to verify critical recommendations.
- **Access Control**: Future work includes authentication to prevent unauthorized access to personal tasks.

7. Testing & Results

- **Unit Tests**: Verified CRUD endpoints, prompt builders, and serialization against Pydantic schemas.
- **Functional Tests**: Manually exercised AI queries, schedule generation, and breakdown suggestions.
- **Performance**: Local inference on an Intel UHD 620 laptop averaged ~1 second per request for 512- token prompts, suitable for interactive use.
- **UI Tests**: Confirmed React forms and alerts render correctly; Dockerized end- to- end smoke tests passed consistently.

8. Challenges & Solutions

- **Model Build Failures**: Required adding `build-essential`, `cmake`, and `libssl-dev` to Dockerfile.
- **Large File Management**: Model weights excluded from Git via `.gitignore` and hosted externally.
- **CORS & Cross- Origin**: Configured FastAPI middleware to allow React dev server on port 3000.

9. Conclusion & Future Work

TaskOptima demonstrates an on- premise AI agent for task management. Future enhancements include: - **User Authentication & Multi- Tenant** support
- **Calendar & Notification** integrations

- **Enhanced Memory**: More nuanced multi- turn dialog state
- **Performance Tuning**: GPU support or higher- performance quantization formats

---