

Local LLM Council

A distributed multi-LLM system where multiple local models collaborate to answer queries, review each other's work, and synthesize a final answer via a Chairman model. Inspired by Andrej Karpathy's LLM Council concept.

Repository: https://github.com/valtpl/LLM_council

Team Information

- **TD Group:** CDOF4
- **Members:**
 - Héloïse ROMEO
 - Valentin TEMPLE
 - Théo RENOIR

Project Overview

This application orchestrates a council of Local LLMs (via Ollama) to provide high-quality answers through a 3-stage process:

1. **Opinions:** Multiple models generate independent answers to a user query.
2. **Peer Review:** Models anonymously review and rank each other's answers for accuracy and insight.
3. **Synthesis:** A designated "Chairman" model analyzes the opinions and reviews to generate a final, comprehensive verdict.

The system supports two deployment modes:

- **Local Mode:** Run all models on a single machine (ideal for testing)
- **Distributed Mode:** Run models across multiple machines on a network (optimal performance)

Setup & Installation

Prerequisites

- **Python 3.8+**
- **Ollama** installed and running on all participating machines.
- Pull the necessary models (e.g., `llama3`, `mistral`, `phi3`):

```
ollama pull llama3
ollama pull mistral
```

1. Create a Virtual Environment

It is recommended to use a virtual environment to manage dependencies.

Windows (PowerShell):

```
# Create the virtual environment
python -m venv venv

# Activate it
.\venv\Scripts\Activate.ps1
# If you get a permission error, run: Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope F
```

Mac/Linux:

```
python3 -m venv venv
source venv/bin/activate
```

2. Install Dependencies

```
pip install -r requirements.txt
```

How to Run

Option A: Local Mode (Single Machine)

Run the entire council on one computer. This is great for testing or if you have a powerful machine.

1. Ensure Ollama is running (`ollama serve` or via the tray icon).

2. Start the app:

```
streamlit run app.py
```

3. In the sidebar, select **Deployment Mode: Local (Single Machine)**.

4. Select the models you want to use for the Council and the Chairman.

5. Click **Initialize Local Council**.

Option B: Distributed Mode (Multiple Machines)

Run the council across multiple computers on the same network (LAN).

1. Configure Member Machines

By default, Ollama only listens to `localhost`. You must enable it to listen to the network on every machine that will act as a Council Member.

On each Member Machine:

1. Stop Ollama if it is running.

2. Open a terminal and run:

Windows (PowerShell):

```
$env:OLLAMA_HOST = "0.0.0.0"  
ollama serve
```

Mac/Linux:

```
OLLAMA_HOST=0.0.0.0 ollama serve
```

3. Find the machine's Local IP address (e.g., `ipconfig` on Windows or `ifconfig` on Mac/Linux).

2. Run the App and Configure Network Members

On the main machine (Chairman):

1. Start the Streamlit application:

```
streamlit run app.py
```

2. In the sidebar, select **Deployment Mode: Distributed (Network)**.

3. Enter the IP addresses and models of your member machines directly in the Streamlit interface:

- **Member Name:** A descriptive name (e.g., "Alex's Laptop")
 - **API URL:** The network address with port (e.g., `http://192.168.1.15:11434`)
 - **Model:** The Ollama model running on that machine (e.g., `llama3`, `mistral`)
4. Add all your council members using the interface.
 5. Click **Initialize Distributed Council** to start the system.

Technical Report

Key Design Decisions

1. Architecture: Council-Based Consensus

The system implements a multi-agent architecture inspired by Andrej Karpathy's LLM Council concept. Key design principles:

- **Three-Stage Pipeline:** The workflow is divided into Opinion Generation, Peer Review, and Chairman Synthesis to ensure comprehensive analysis
- **Concurrent Execution:** Uses Python's `concurrent.futures.ThreadPoolExecutor` to parallelize API calls to different models, significantly reducing response time
- **Flexible Deployment:** Supports both local and distributed modes to accommodate different hardware configurations

2. Error Handling & Health Monitoring

- **Health Checks:** Before running queries, the system pings all nodes to verify availability
- **Timeout Management:** Extended timeout (600s) for model generation to handle complex queries
- **Graceful Degradation:** If a model fails to respond, the system continues with available models rather than failing completely

3. User Interface

- **Streamlit Framework:** Chosen for its simplicity and real-time feedback capabilities
- **Dynamic Model Selection:** The UI automatically detects available Ollama models in both modes
- **Interactive Configuration:** In distributed mode, users can configure member machines directly through the web interface without editing code

Chosen LLM Models

The system is model-agnostic and works with any models available through Ollama. Default configuration includes:

1. Council Members:

- **Llama 3.2 (1B)**: Lightweight model providing quick responses with good general knowledge
- **Mistral**: Strong reasoning capabilities and concise answers
- **Phi-3**: Microsoft's efficient model with excellent instruction-following

2. Chairman:

- **Mistral 7B**: Chosen for synthesis due to its superior reasoning capabilities, strong performance in aggregating multiple perspectives, and excellent ability to process longer context from multiple opinions

Model Selection Rationale:

- **Diversity**: Different model architectures (Meta, Mistral AI, Microsoft) provide varied perspectives
- **Resource Efficiency**: 1B and small models allow running on consumer hardware
- **Customizable**: Users can select any Ollama-compatible models through the UI

Improvements Over Original Repository

This implementation builds upon the LLM Council concept with several enhancements:

1. Dual Deployment Mode:

- Added local mode for single-machine testing
- Maintained distributed mode for network deployment
- Dynamic switching between modes without code changes

2. Interactive Web Interface:

- Replaced command-line interface with user-friendly Streamlit UI
- Real-time progress tracking during the three stages
- Visual feedback with expandable sections for opinions and reviews

3. Better Configuration Management:

- Centralized configuration in `config.py`
- Dynamic model discovery in local mode
- Clear separation between local and network settings

4. Enhanced Robustness:

- Health monitoring system to detect unavailable nodes
- Increased timeouts for more complex queries
- Better error messages and user feedback

5. Concurrent Processing:

- Parallel API calls to all models simultaneously
- Significant performance improvement over sequential execution
- Efficient resource utilization

6. Improved Prompt Engineering:

- Structured prompts for each stage (opinion, review, synthesis)
- System prompts tailored to each role (expert, reviewer, chairman)
- Better context management for the Chairman's synthesis

Generative AI Usage & Penalties Statement

Declaration of Generative AI Usage

This project was developed with the assistance of Generative AI tools, in full compliance with academic guidelines. We declare the following usage:

Tools Used:

- **GitHub Copilot (Claude Sonnet 4.5)** - Primary AI assistant
- **Gemini** - Supplementary research and clarification

Purpose & Methodology:

1. Initial Understanding & Planning (15% of work)

- Used AI to analyze and break down project requirements
- Clarified the LLM Council concept and architectural patterns
- Explored different implementation strategies and best practices

2. Code Development (60% of work)

- **Core Architecture:** AI-assisted implementation of the three-stage pipeline (opinion generation, peer review, synthesis)
- **Concurrent Processing:** Guidance on using `ThreadPoolExecutor` for parallel API calls
- **API Integration:** Assistance with Ollama API communication and error handling
- **UI Development:** Streamlit interface scaffolding and real-time feedback implementation
- **Configuration Management:** Design of flexible local/distributed deployment system

3. Debugging & Optimization (15% of work)

- Troubleshooting network connectivity issues in distributed mode
- Refining timeout handling and health check mechanisms
- Optimizing prompt engineering for better model responses

4. Documentation (10% of work)

- Structuring the README with clear setup instructions
- Writing technical explanations of design decisions
- Creating comprehensive user guides for both deployment modes

Our Contribution:

- Final design decisions and architectural choices
- Testing across local and distributed environments
- Network configuration and deployment validation
- Integration and refinement of AI-generated code
- Quality assurance and user experience optimization