

# Mini report - Local Deployment of an LLM Council

*Work done by COCURON Morgane and KATIBI Maria (CDOF2)*

## Introduction

This project aims to implement a fully local LLM council inspired by the concept introduced by Andrej Kerpathy.

Instead of relying on a simple large language model, the system coordinates multiple LLMs that collaborate to answer a user query, critique each other's responses, and produce a final synthesized answer through a dedicated model called the Chairman.

Unlike the original cloud-based implementation, this project enforces a fully local deployment, without any external APO or cloud service, relying instead on local models and REST API communication between machines.

## LLM Council Principals

The LLM Council operates in three successive stages.

First, multiple language models independently generate responses to the same user query. In the second stage, each model anonymously reviews and ranks the responses produced by the others.

Finally, a dedicated model called the Chairman aggregates all responses and rankings to generate a single, high-quality final answer.

## System Architecture

The system is based on a distributed architecture involving two separate machines.

The first machine (PC A) acts as the orchestrator. It hosts the FastAPI backend, the LLM Council logic, and multiple local models responsible for generating initial responses and reviews.

The second machine (PC B) is dedicated exclusively to the Chairman role. It runs Ollama and exposes a single language model through a REST API, without hosting any project code.

Communication between the two machines occurs through the local network, ensuring a realistic deployment scenario with explicit inter-machine API calls.

```
# Council = sur TON PC (au début) donc localhost.  
# En démo binôme: Council reste sur ton PC -> localhost côté orchestrator si orchestrator est sur ton PC.  
COUNCIL_BASE_URL = os.getenv("COUNCIL_BASE_URL", "http://localhost:11434")  
  
# Chairman = au début localhost, puis en démo: http://IP_PC_COPINE:11434  
CHAIRMAN_BASE_URL = os.getenv("CHAIRMAN_BASE_URL", "http://localhost:11434")
```

## Modifications from the original project

The original project relied on OpenRouter as a cloud-based hub to access multiple LLMs. In this implementation, OpenRouter was replaced by Ollama in order to execute all models locally.

To support the fully local deployment, we updated the `.env` files for both the backend and the frontend.

For the **backend**, the `.env` file defines the Ollama endpoints and models used by the system. `COUNCIL_BASE_URL` points to the local Ollama server hosting the Council models (mistral, qwen2.5, llama3), while `CHAIRMAN_BASE_URL` targets the second machine on the local network for the Chairman model. It also includes parameters such as parallel requests and retry settings to improve stability.

For the **frontend**, the `.env` file sets the API base URL (`VITE_API_BASE`) to ensure the interface communicates correctly with the FastAPI backend.

```
backend > ⚙ .env
 1  # === Council LLMs (sur mon PC) ===
 2  COUNCIL_BASE_URL=http://localhost:11434
 3
 4  COUNCIL_MODEL_1=mistral
 5  COUNCIL_MODEL_2=qwen2.5
 6  COUNCIL_MODEL_3=llama3
 7
 8  # === Chairman (IP du PC du binôme) ===
 9  CHAIRMAN_BASE_URL=http://172.20.10.4:11434
10  CHAIRMAN_MODEL=llama3
11
12  MAX_PARALLEL_REQUESTS=1
13  RETRY_COUNT=1
14  RETRY_SLEEP_SECONDS=0.8
15

frontend > ⚙ .env
 1  # URL de base de l'API backend (FastAPI)
 2  VITE_API_BASE=http://localhost:8001
```

A dedicated client was implemented to interact with Ollama's REST API, while configuration files were updated to define local models and network endpoints.

Importantly, the core logic of the LLM Council remained unchanged; only the infrastructure and communication layers were adapted to support local deployment.

More specifically, we replaced the original file called `openrouter.py` with a dedicated file named `ollama_client.py`, responsible for handling all interactions with locally deployed models.

```
"""Ollama API client for making LLM requests."""

import httpx
from typing import List, Dict, Any, Optional

async def ollama_chat(
    base_url: str,
    model: str,
    messages: List[Dict[str, str]],
    timeout: float = 420.0,
) -> Optional[str]:
    """
    Appelle l'API Ollama /api/chat et retourne le texte assistant.
    Doc officielle: POST /api/chat. (cf docs Ollama)
    """

    url = f"{base_url.rstrip('/')}api/chat"
    # Payload conforme à l'API Ollama
    payload = {
        "model": model,
        "messages": messages,
        "stream": False,
    }

    async with httpx.AsyncClient(timeout=timeout) as client:
        r = await client.post(url, json=payload)
        r.raise_for_status()
        data = r.json()

    return data.get("text")
```

This screenshot illustrates the implementation of a new client designed to replace the original OpenRouter API client.

Instead of sending requests to a cloud-based hub, this client communicates directly with a locally deployed Ollama server through its REST endpoint `/api/chat`.

The function `ollama_chat` is implemented asynchronously to ensure compatibility with FastAPI and to allow non-blocking calls to language models. It sends a list of messages following an OpenAI-compatible chat format and retrieves only the generated assistant response from the Ollama server.

By abstracting the model interaction behind a dedicated client, the existing LLM Council logic remains unchanged. This design choice allows the infrastructure layer to be modified without impacting the core multi-LLM orchestration mechanism.

```

d > config.py > ...
"""
Configuration for the LLM Council.
Ce fichier centralise toute la configuration liée :
- aux endpoints Ollama (Council / Chairman)
- aux modèles utilisés
- à la gestion de la concurrence et de la stabilité
"""
import os
from dotenv import load_dotenv
|
# Charge les variables d'environnement depuis un fichier .env
load_dotenv()

# Le conseil reste sur mon PC
COUNCIL_BASE_URL = os.getenv("COUNCIL_BASE_URL", "http://localhost:11434")

# Chairman tourne sur le pc de l'autre membre du binôme. On utilise l'IP du second ordinateur
CHAIRMAN_BASE_URL = os.getenv("CHAIRMAN_BASE_URL", "http://172.20.10.4:11434")

# Liste des modèles composant le Council
# Chaque modèle peut être surchargé via les variables d'environnement
COUNCIL_MODELS = [
    os.getenv("COUNCIL_MODEL_1", "llama3"),
    os.getenv("COUNCIL_MODEL_2", "mistral"),
    os.getenv("COUNCIL_MODEL_3", "qwen2.5"),
]
# Modèle utilisé par le Chairman pour la synthèse / décision finale
CHAIRMAN_MODEL = os.getenv("CHAIRMAN_MODEL", "llama3")

# Répertoire où sont sauvegardées les conversations
DATA_DIR = "data/conversations"

# Nombre maximum de requêtes parallèles envoyées aux modèles
# 1 = mode safe
MAX_PARALLEL_REQUESTS = int(os.getenv("MAX_PARALLEL_REQUESTS", "1")) # 1 = safe

# Nombre de tentatives en cas d'échec d'une requête

```

This screenshot presents the updated configuration config.py file used in the local deployment of the LLM Council.

Unlike the original cloud-based version, which relied on OpenRouter endpoints and proprietary model identifiers, this configuration defines fully local endpoints and models executed through Ollama.

The `COUNCIL_BASE_URL` parameter specifies the address of the Ollama server hosting the council models on the orchestrator machine, while the `CHAIRMAN_BASE_URL` parameter can be dynamically updated to point to a remote machine during the live demonstration. This enables real network communication between the orchestrator and the Chairman model.

The list of `COUNCIL_MODELS` now contains locally deployed Ollama models, such as `llama3`, `mistral`, and `qwen2.5`, replacing all cloud-based LLM providers. The Chairman model is similarly defined as a local Ollama model.

These modifications are strictly limited to configuration parameters, ensuring that the core logic and orchestration flow of the LLM Council remain unchanged while enabling a fully local and distributed execution.

The LLM Council is composed of three locally deployed models: *llama3*, *mistral*, and *qwen2.5*. These models were selected to provide architectural and training diversity within the council, allowing different reasoning styles and response patterns.

The Chairman role is assigned to *llama3* due to its strong balance between reasoning capabilities, response coherence, and stability during longer context aggregation. Compared to other council models, *llama3* demonstrates more consistent behavior when synthesizing multiple inputs into a single structured response, which is a critical requirement for the final decision-making stage of the LLM Council.

After adapting the infrastructure, the main objective was to ensure that the **three-stage orchestration** of the council could run entirely locally via Ollama. This orchestration is implemented in the `council.py` file, which contains the full pipeline logic:

- **Stage 1 – Response collection:** each model in the council answers the user query independently. Requests are executed asynchronously with a semaphore mechanism to limit the number of parallel calls and avoid overloading Ollama.
- **Stage 2 – Peer review and ranking:** each model acts as a reviewer and evaluates anonymized responses. Responses are labeled as *Response A*, *Response B*, *Response C*, etc., ensuring that reviewers cannot bias their ranking based on model identity. A strict ranking format is enforced, and the ranking is parsed automatically for aggregation.
- **Stage 3 – Final synthesis by the Chairman:** the Chairman receives the original question, all Stage 1 responses, and the Stage 2 peer rankings. It then produces a single final answer, resolving disagreements when possible or explicitly mentioning uncertainty.

## Why Ollama?

Ollama was chosen because it provides a simple and efficient way to run LLMs fully locally, without relying on any cloud API. It exposes models through a REST interface, which perfectly matches our distributed architecture (PC A orchestrator + PC B Chairman). Ollama is also easy to deploy, lightweight, and supports multiple open-source models with consistent request formatting.

## Why **llama3**, **Qwen2.5** and **Mistral** as LLM ?

These three models were selected among Ollama's available models because they offer a good mix between quality, speed, and diversity:

- *Llama3* was selected for its strong reasoning capabilities and stable behavior, especially for long-context synthesis, which is required for the Chairman role.
- *Mistral* is fast and efficient while still producing high-quality answers, making it suitable for both response generation and peer-review.

- Qwen2.5 brings a different training background and output style, which increases diversity in the council and improves robustness when models disagree.

Other Ollama models were not selected mainly due to higher hardware requirements, less stable behavior during long prompts, or weaker performance in structured ranking and synthesis tasks. This combination was therefore the most reliable for a local multi-model pipeline under limited compute resources.

Another important factor in choosing **llama3**, **mistral**, and **qwen2.5** is that each model remains reasonably lightweight for local deployment. In our setup, downloading and installing each model required around **4 GB**, which is fully manageable on our machines and allows the full council to run locally without excessive storage constraints.

## **Deployment et execution**

The backend is implemented using FastAPI, which serves as the local orchestration layer for the LLM Council.

Once the server is running, it sequentially triggers the three stages of the council process and communicates with the Chairman model through HTTP requests.

Ollama is used on both machines to serve local language models via a REST API, enabling seamless inter-process and inter-machine communication

## **Testing and Validation**

To validate the correct behavior of the local LLM Council, several tests were performed using the web interface connected to the FastAPI backend.

First, we verified the full 3-stage pipeline execution on a simple user query (“What is deep learning?”). The results confirm that Stage 1 successfully collects independent responses from the council models (llama3, mistral, qwen2.5). Stage 2 then performs the peer-review process by anonymizing the answers (Response A, B, C) and producing a strict final ranking. Finally, Stage 3 generates a single final answer through the Chairman model, synthesizing the council responses and the peer rankings into a structured output.

YOU

qu'est ce que le deep learning ?

LLM COUNCIL

#### Stage 1: Individual Responses

llama3    mistral    qwen2.5

llama3

Le deep learning (ou apprentissage profond) est une branche de l'intelligence artificielle qui se concentre sur les réseaux de neurones artificiels composés de plusieurs couches de traitement de l'information. Ces réseaux peuvent apprendre automatiquement à partir d'un grand nombre de données et identifier des modèles complexes dans ces données.

Les réseaux de neurones profonds peuvent être entraînés pour accomplir diverses tâches, telles que :

- La reconnaissance des formes (par exemple, la reconnaissance d'images) ;
- Le traitement du langage naturel (par exemple, la compréhension de textes) ;
- La génération de texte ou de sons ;
- L'apprentissage de mouvements et de comportements.

Le deep learning est souvent utilisé dans des domaines tels que l'intelligence artificielle, le traitement du langage naturel, les réseaux sociaux, la reconnaissance des formes, la biologie computationnelle, etc.

#### Stage 2: Peer Rankings

##### Raw Evaluations

Each model evaluated all responses (anonymized as Response A, B, C, etc.) and provided rankings. Below, model names are shown in **bold** for readability, but the original evaluation used anonymous labels.

llama3    mistral    qwen2.5

llama3

Here is my evaluation and ranking:

Evaluation:

Response A: Strengths: Clear and concise definition of deep learning, provides examples of its applications. Weaknesses: Could be more precise in defining the characteristics of deep learning.

Response B: Strengths: Provides a good overview of what deep learning can do, mentions the statistical learning models used. Weaknesses: The text is a bit vague and could be more specific about how deep learning works.

Response C: Strengths: Gives an idea of the inspiration behind deep learning, which is interesting. Weaknesses: Does not provide enough information about how deep learning actually works or its applications.

FINAL RANKING:

1. Response A
2. Response B
3. Response C

---

##### Extracted Ranking:

1. Response A
2. Response B

### Stage 3: Final Council Answer

Chairman: llama3

Le deep learning (ou apprentissage profond) est une branche de l'intelligence artificielle qui se concentre sur les réseaux de neurones artificiels composés de plusieurs couches de traitement de l'information. Ces réseaux peuvent apprendre automatiquement à partir d'un grand nombre de données et identifier des modèles complexes dans ces données.

Les réseaux de neurones profonds peuvent être entraînés pour accomplir diverses tâches, telles que la reconnaissance des formes (par exemple, la reconnaissance d'images), le traitement du langage naturel (par exemple, la compréhension de textes), la génération de texte ou de sons, et l'apprentissage de mouvements et de comportements.

Le deep learning est souvent utilisé dans des domaines tels que l'intelligence artificielle, le traitement du langage naturel, les réseaux sociaux, la reconnaissance des formes, la biologie computationnelle, etc.

(Note: After reviewing the individual responses and peer rankings, I have consolidated the answers to provide a clear and structured final answer that reflects the majority consensus. Any disagreements or uncertainties are minimal and do not significantly impact the overall understanding of deep learning.)

In a second test, we intentionally simulated a failure scenario by disconnecting the Chairman machine. This was done by closing the terminal window running the Ollama service (e.g., the process started with ollama serve) on PC B. As expected, the backend was no longer able to reach the Chairman endpoint, and the system returned the error "ConnectError: All connection attempts failed". This confirms that the distributed architecture relies on real inter-machine REST communication and that the system correctly handles network/service interruptions by returning a clear error message instead of crashing.

Chairman: llama3

#### Extracted Ranking:

1. Response A
2. Response B
3. Response C

### Stage 3: Final Council Answer

Chairman: llama3

Error: Chairman failed: ConnectError: All connection attempts failed

## Conclusion

This project demonstrates the feasibility of deploying a collaborative multi-LLM system entirely in a local environment.

By combining FastAPI, Ollama, and a distributed architecture, the LLM Council replicates the behavior of cloud-based multi-model systems while remaining fully self-hosted.

The project highlights both the advantages and technical challenges of local LLM deployment and provides a strong foundation for more advanced multi-agent AI systems.

In addition, the local implementation improves the original repository in several key aspects: **privacy and autonomy** by removing any external API dependency, a more **realistic deployment** through true inter-machine REST communication, and increased **stability** thanks to the retry mechanism and the parallel request limiting system.

**NB:** The environment setup and configuration steps are described in the project GitHub README file.

**Github project link :** <https://github.com/morganecocuron/GENAI-Projet-COCURON-KATIBI>