Rapport Technique - LLM Council Local Deployment

Projet : Local LLM Consensus Engine
Équipe : Tom Delahaye, Pierre Briand
Date : Janvier 2026
Repo Github : https://github.com/Tom-dlhy/local-llm-consensus-engine

Table des Matières

1. Introduction

Ce projet implémente le concept **"LLM Council"** d'Andrej Karpathy en version **100% locale et distribuée**. Au lieu de s'appuyer sur des API cloud (OpenRouter), notre système utilise **Ollama** pour exécuter plusieurs LLMs localement sur deux machines communiquant via API REST.

Objectifs Atteints

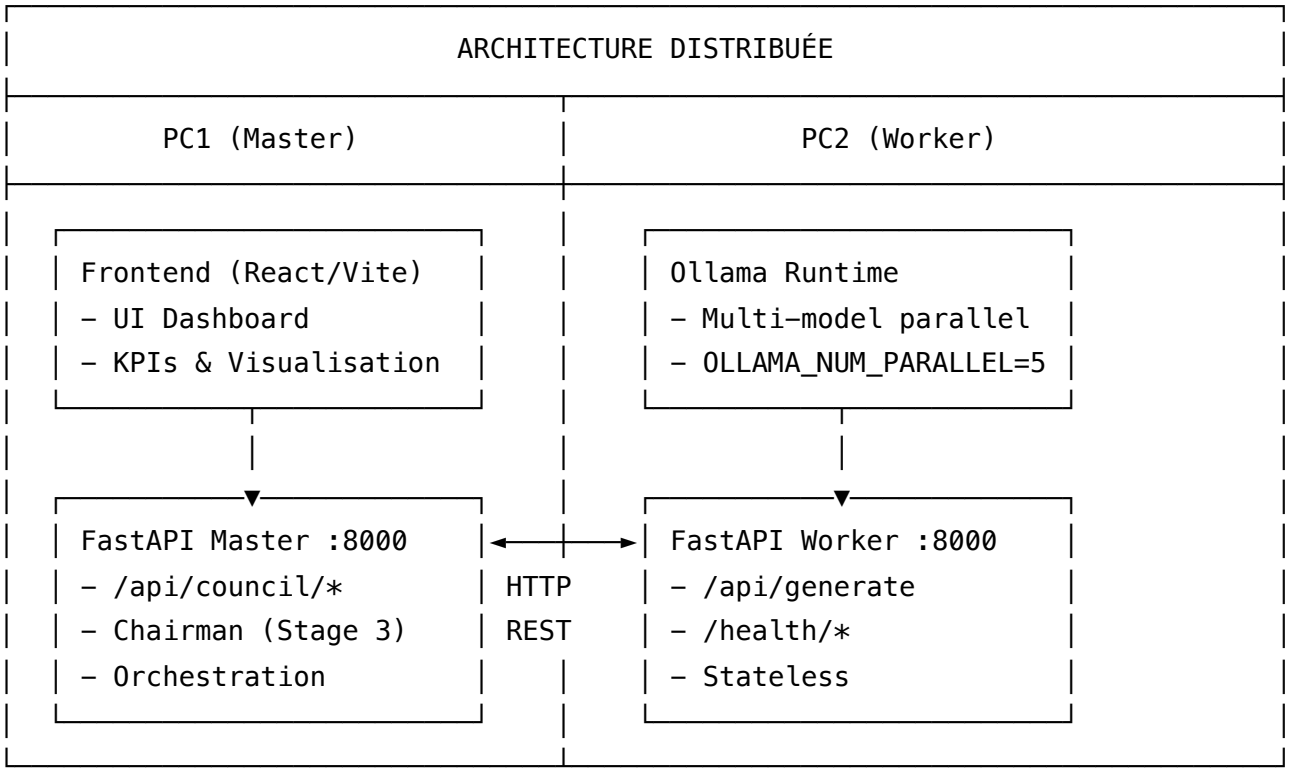
- Exécution locale complète (aucune dépendance cloud)
- Architecture distribuée Master/Worker sur 2 PCs
- Chairman séparé sur PC1 (synthèse finale)

- Interface utilisateur moderne avec monitoring en temps réel

2. Architecture Système

2.1 Architecture Distribuée PC1/PC2

Nous avons implémenté un pattern **Master/Worker** distribué :



2.2 Justification de l'Architecture

Aspect	Choix	Justification
Pattern	Master/Worker	Séparation claire des responsabilités
Chairman	Sur PC1 (Master)	Respecte la contrainte du sujet
Communication	HTTP REST	Simple, fiable, compatible réseau local
Parallélisme	asyncio.gather	Exécution simultanée de tous les agents

2.3 Rôle du Chairman

Le Chairman est une **responsabilité du Master** (PC1), pas un service séparé. Cette approche :

- Simplifie l'architecture
- Respecte la contrainte "Chairman sur PC #1"
- Permet au Chairman d'accéder directement à Ollama local pour la synthèse

3. Stack Technique Backend

3.1 Technologies Utilisées

Composant	Technologie	Version	Justification
Langage	Python	3.12+	Async natif, typage fort
Framework API	FastAPI	≥0.115	Async, WebSocket, OpenAPI auto
Configuration	Pydantic Settings	≥2.6	Validation, type safety, env vars
HTTP Client	httpx	≥0.27	Async, timeouts configurables
Monitoring	psutil	≥6.0	CPU/RAM metrics, cross-platform
LLM Runtime	Ollama	Latest	Simple, multi-model, API REST
Package Manager	uv	Latest	Rapide, lock files

3.2 Structure du Code Backend

```
backend/
├── src/
│   ├── main.py              # Point d'entrée, configuration FastAPI
│   ├── config.py            # Settings Pydantic (Role, URL, Timeouts)
│   ├── api/
│   │   ├── council_routes.py   # Endpoints Master (/api/council/*)
│   │   ├── worker_routes.py    # Endpoints Worker (/api/generate)
│   │   └── health_routes.py    # Health checks (/health/*)
│   ├── models/
│   │   └── council.py          # Modèles Pydantic (Session, Agent, Review)
│   └── services/
│       ├── council.py          # Orchestration 3-stage workflow
│       └── ollama_client.py    # Client HTTP Ollama
├── pyproject.toml           # Dépendances Python
└── .env                     # Configuration environnement
```

3.3 Gestion des Timeouts

Problème : Les LLMs peuvent prendre 30-60s par réponse.

Solution :

```
httpx.Timeout(120.0, connect=10.0)  # 120s total, 10s connect
```

3.4 Parallélisme avec asyncio

```
# Stage 1: Toutes les opinions en parallèle
tasks = [self._generate_opinion(agent) for agent in agents]
results = await asyncio.gather(*tasks, return_exceptions=True)
```

Avec `OLLAMA_NUM_PARALLEL=5` sur PC2, les 5 modèles génèrent simultanément.

3.5 API Endpoints

Worker (PC2)

Endpoint	Méthode	Description
`/api/generate`	POST	Génération LLM simple
`/health`	GET	Statut du service
`/health/models`	GET	Modèles Ollama disponibles

Master (PC1)

Endpoint	Méthode	Description
`/api/council/query`	POST	Lancer une délibération
`/api/council/session/{id}`	GET	État d'une session
`/api/council/models`	GET	Modèles recommandés
`/api/council/ws/{id}`	WebSocket	Streaming temps réel

4. Stack Technique Frontend

4.1 Technologies Utilisées

Composant	Technologie	Version	Justification
Framework	React	19.0	Composants, hooks, state moderne
Routing	TanStack Router	1.136	Type-safe, SSR ready
Build	Vite	7.1	HMR rapide, bundling optimisé
UI Components	shadcn/ui + Radix	Latest	Accessibles, stylés Tailwind
Styling	Tailwind CSS	4.1	Utility-first, dark mode natif
Charts	Recharts	3.6	Graphiques React natifs

Composant	Technologie	Version	Justification
Icons	Lucide React	0.554	Icons modernes, tree-shakable

4.2 Structure du Code Frontend

```
frontend/src/
├── components/
│   ├── council/              # Composants métier
│   │   ├── ModelSelector.tsx    # Sélection des modèles
│   │   ├── QueryForm.tsx        # Formulaire de question
│   │   ├── StageProgress.tsx    # Indicateur de progression
│   │   ├── OpinionCard.tsx      # Opinion agent (Stage 1)
│   │   ├── ReviewCard.tsx       # Review avec scores (Stage 2)
│   │   ├── FinalAnswerCard.tsx  # Réponse Chairman (Stage 3)
│   │   └── TokenUsageStats.tsx  # Statistiques tokens
│   └── ui/                   # Composants shadcn/radix
├── routes/                   # Pages (/, /responses, /kpis)
├── services/                 # API clients
└── types/                    # Types TypeScript (miroir backend)
```

4.3 Pages Principales

Page	URL	Fonctionnalité
Chat	`/`	Sélection modèles, question, réponse finale
Responses	`/responses`	Détail des opinions et reviews
KPIs	`/kpis`	Dashboard métriques et graphiques

5. Workflow du Council (3 Stages)

Stage 1 : First Opinions

1. L'utilisateur soumet une question
2. Tous les agents LLM génèrent une réponse **en parallèle**
3. Chaque réponse est stockée avec ses métriques (tokens, durée)

```
responses = await asyncio.gather(*[generate(agent) for agent in agents])
```

Stage 2 : Review & Ranking (Pairwise)

1. Chaque agent évalue les réponses des **autres** agents (pas la sienne)
2. Pour N agents, cela génère N×(N-1) évaluations parallèles
3. Scores de 1 à 10 avec justification en JSON

Exemple avec 3 agents :

- Agent 1 évalue Agent 2 et Agent 3
- Agent 2 évalue Agent 1 et Agent 3
- Agent 3 évalue Agent 1 et Agent 2
- = 6 évaluations parallèles

Stage 3 : Chairman Synthesis

1. Le Chairman (PC1) reçoit toutes les opinions + classements
2. Il synthétise une réponse finale unique
3. Les sources les mieux notées sont mises en avant

6. Modèles LLM Utilisés

6.1 Modèles Recommandés

Modèle	Taille	Rôle Optimal	Justification
`qwen2.5:0.5b`	350 MB	Stage 1 (Opinions)	Rapide, léger
`llama3.2:1b`	1.3 GB	Stage 2 (Review)	Bon équilibre
`gemma2:2b`	1.6 GB	Agent expert	Précis
`phi3.5:latest`	2.2 GB	Chairman (Stage 3)	Synthèse qualité
`tinyllama`	600 MB	Backup	Ultra léger

6.2 Configuration Ollama (PC2)

```
export OLLAMA_NUM_PARALLEL=5       # 5 requêtes simultanées
export OLLAMA_MAX_LOADED_MODELS=5  # 5 modèles en mémoire
ollama serve
```

7. Améliorations et Bonus

7.1 Monitoring Avancé

Feature	Description	Implémentation
Token Usage	Suivi prompt/completion par stage	`SessionTokenUsage` model
Latency Tracking	Temps par modèle et end-to-end	`SessionLatencyStats` model
Health Checks	CPU, RAM, Ollama status	`/health/*` endpoints
Model Status	Liste des modèles disponibles	`/health/models`

7.2 UI/UX Améliorations

Feature	Description
Dark Mode	Thème sombre natif via Tailwind
Stage Progress	Indicateur visuel de progression
Code Couleur	Couleurs distinctes par modèle
Responsive	Interface adaptée mobile/desktop

7.3 Visualisations Avancées

Graphique	Page	Description
Pie Chart	`/kpis`	Distribution tokens par modèle
Bar Chart	`/kpis`	Latence par modèle (ms)

Graphique	Page	Description
Radar Charts	`/kpis`	Performance multi-dimensionnelle par modèle

7.4 Métriques Exemple (3 agents)

Stage	Prompt Tokens	Completion Tokens	Total
Stage 1 (Opinions)	244	256	500
Stage 2 (Review)	1,293	594	1,887
Stage 3 (Synthesis)	501	352	853
TOTAL	**2,038**	**1,202**	**3,240**

8. Installation et Déploiement

8.1 Prérequis

- Python 3.12+
- Node.js 18+
- uv (package manager Python)
- Ollama

8.2 Installation des Modèles

```
ollama pull qwen2.5:0.5b
ollama pull llama3.2:1b
ollama pull gemma2:2b
ollama pull phi3.5:latest
ollama pull tinyllama
```

8.3 Déploiement Distribué

PC2 (Worker)

```
export OLLAMA_NUM_PARALLEL=5
export OLLAMA_MAX_LOADED_MODELS=5
ollama serve

# Nouveau terminal
cd backend
uv sync
uv run python -m src.main --role worker --host 0.0.0.0 --port 8000
```

PC1 (Master)

```
cd backend
uv sync
uv run python -m src.main --role master --worker-url http://IP_PC2:8000

# Frontend (nouveau terminal)
cd frontend
npm install
npm run dev
```

8.4 Vérification

```
# Test PC2
curl http://IP_PC2:8000/health/models

# Test PC1
curl http://localhost:8000/health
```

9. Conclusion

Ce projet implémente avec succès le concept LLM Council en version **locale et distribuée**. Les principales réalisations sont :

Objectifs Atteints

- **Exécution locale** : Ollama remplace les API cloud
- **Architecture distribuée** : Master/Worker sur 2 PCs
- **Chairman séparé** : Synthèse sur PC1 uniquement
- **Workflow complet** : 3 stages fonctionnels

Améliorations Bonus

- Monitoring complet (tokens, latence, health)
- Dashboard KPIs avec graphiques
- Interface moderne (dark mode, responsive)
- Évaluations pairwise (N×(N-1) reviews parallèles)

Points Techniques Clés

- Parallélisme avec `asyncio.gather`
- JSON mode pour reviews structurés
- WebSocket pour streaming temps réel
- Types TypeScript miroir des modèles Pydantic

Annexes

Déclaration d'Usage d'IA Générative

Déclaration

Conformément aux exigences du projet, je déclare avoir utilisé des outils d'Intelligence Artificielle Générative dans le cadre du développement de ce projet.

Outils Utilisés

Outil	Version/Modèle
Claude (Anthropic)	Claude 4.5 Opus

Usages Détaillés

1. Documentation

Fichiers concernés :

- `README.md` - Documentation principale du projet
- `backend/README.md` - Documentation du backend
- `project/backend-project.md` - Justifications techniques backend
- `project/frontend-project.md` - Justifications techniques frontend
- `DECLARATION.md` - Ce fichier

But : Rédaction et structuration de la documentation technique, incluant les instructions d'installation, les guides de démarrage, et les explications d'architecture.

2. Logging et Configuration

Fichiers concernés :

- `backend/.env` - Configuration environnement
- `backend/.env.example` - Template de configuration

But : Génération des fichiers de configuration avec commentaires explicatifs et structuration des variables d'environnement.

3. Adaptation du Frontend

Fichiers concernés :

- `frontend/` - Ensemble des composants React/Vite

But : Adaptation et personnalisation d'un frontend existant provenant d'un autre projet personnel pour l'intégrer au système LLM Council. Cela inclut :

- Modification des composants UI
- Adaptation des appels API
- Personnalisation du thème et des styles

4. Refactorisation et Optimisation

Fichiers concernés :

- `backend/src/` - Services et routes du backend
- `frontend/src/` - Composants React

But : Refactorisation du code existant pour améliorer les performances et la maintenabilité :

- Optimisation des appels asynchrones
- Réduction de la duplication de code
- Amélioration de la gestion des erreurs
- Simplification des structures de données

Code Non Assisté par IA

Les éléments suivants ont été développés **sans assistance d'IA générative** :

- Architecture distribuée Master/Worker
- Logique d'orchestration du Council (Stage 1-2-3)
- Algorithme de consensus et scoring
- Intégration Ollama