AYI NEDJIMI Consultants

LLM en Local : Ollama, LM Studio et vLLM - Comparatif 2026

Catégorie : Intelligence Artificielle Lecture : 14 min Publié le : 13/02/2026 Auteur : Ayi NEDJIMI

Comparatif complet Ollama vs LM Studio vs vLLM pour exécuter des LLM en local. Installation, performances, cas d'usage et guide de choix 2026.

LLM en Local : Ollama, LM Studio et vLLM - Comparatif 2026 constitue un enjeu majeur pour les professionnels de la sécurité informatique et les équipes techniques. Ce guide détaillé sur ia llm local ollama lmstudio propose une méthodologie structurée, des outils éprouvés et des recommandations opérationnelles directement applicables. L'objectif est de fournir aux praticiens — consultants, ingénieurs sécurité, administrateurs systèmes — les connaissances et les techniques nécessaires pour aborder ce sujet avec rigueur. Chaque section s'appuie sur des retours d'expérience terrain et intègre les évolutions les plus récentes du domaine. Les recommandations présentées sont adaptées aux environnements d'entreprise et tiennent compte des contraintes opérationnelles réelles.

Table des Matières

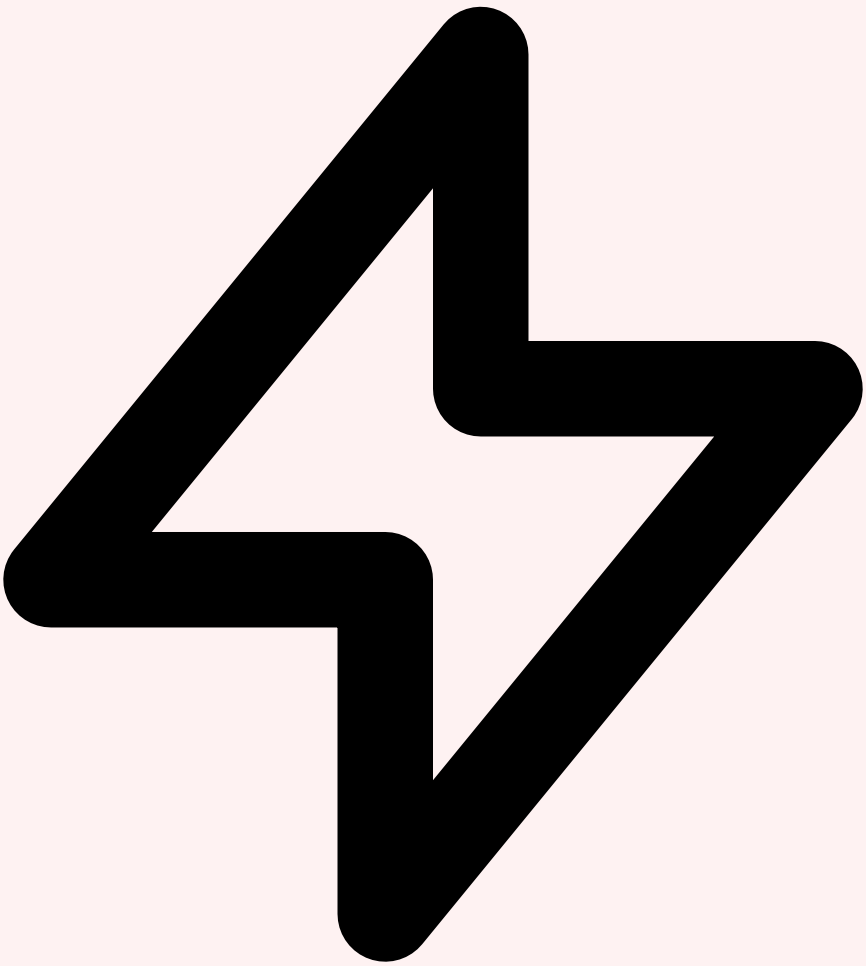
Notre avis d'expert

Chez Ayi NEDJIMI Consultants, nous constatons que la majorité des organisations sous-estiment les risques liés aux modèles de langage déployés en production. La sécurité des LLM ne se limite pas au prompt engineering : elle exige une approche systémique couvrant les embeddings, les pipelines de données et les mécanismes de contrôle d'accès aux API. Comparatif complet Ollama vs LM Studio vs vLLM pour exécuter des LLM en local. Installation, performances, cas d'usage et guide de choix 2026. Dans un contexte où l'intelligence artificielle transforme les pratiques de cybersécurité, la maîtrise de ia llm local ollama lmstudio devient un avantage stratégique pour les équipes techniques. Nous abordons notamment : table des matières, 1. pourquoi exécuter un llm en local ? et 2. ollama : la simplicité au service du llm local. Les professionnels y trouveront des recommandations actionnables, des commandes prêtes à l'emploi et des stratégies de mise en œuvre adaptées aux environnements d'entreprise.

Votre organisation est-elle prête à faire face aux attaques basées sur l'IA ?

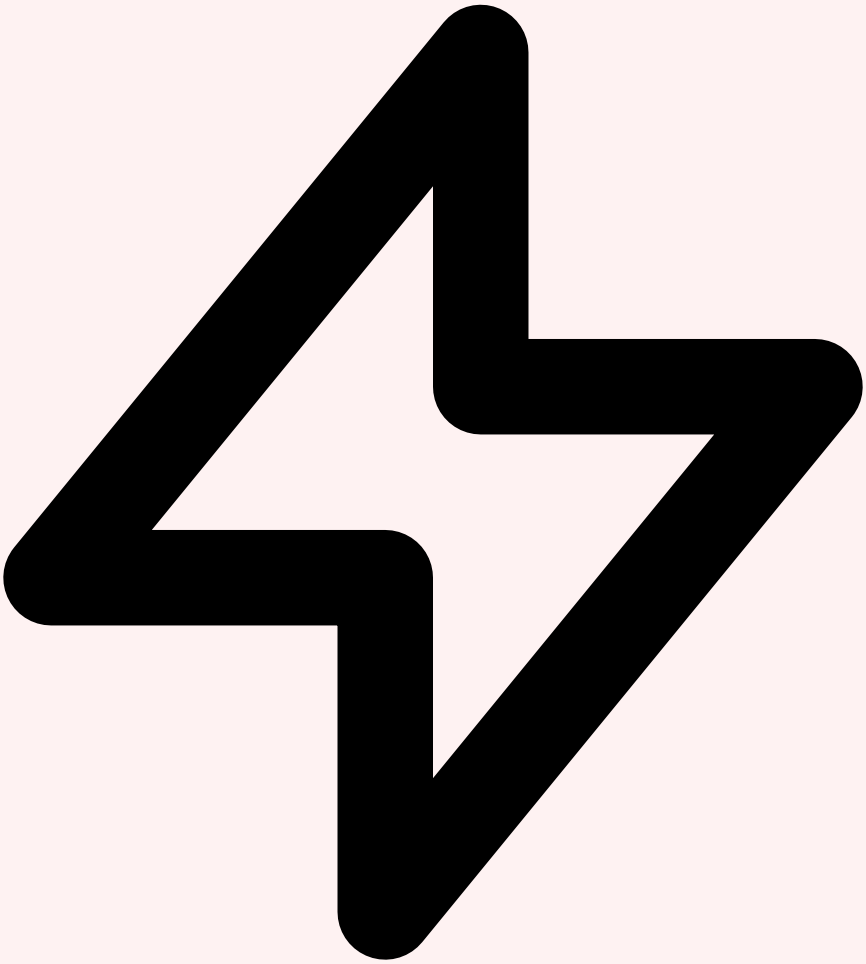
1. Pourquoi Exécuter un LLM en Local ?

L'exécution de modèles de langage en local constitue une tendance majeure de l'année 2026. Alors que les API cloud (OpenAI, Anthropic, Google) dominent le marché grand public, de plus en plus d'entreprises et de développeurs choisissent de faire tourner leurs propres modèles sur leur infrastructure. Les raisons de cette transition sont multiples et souvent complémentaires.



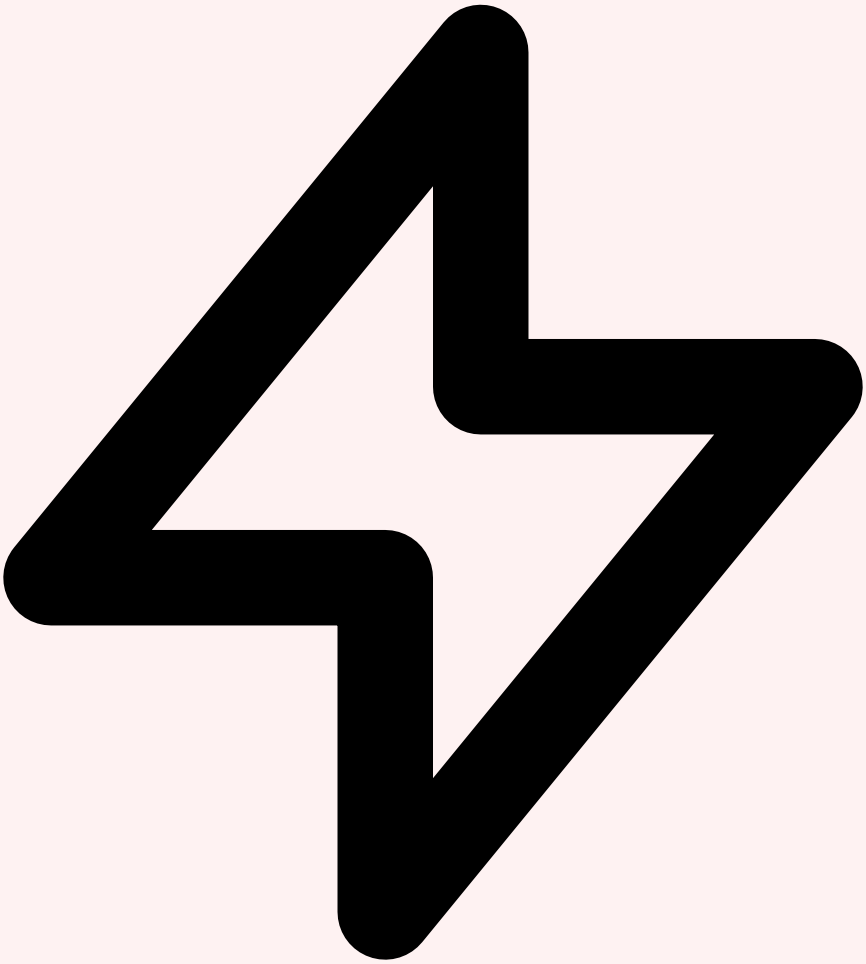
Confidentialité et Souveraineté des Données

L'argument le plus puissant en faveur du LLM local reste la **confidentialité des données**. Lorsque vous envoyez une requête à une API cloud, vos données transitent par des serveurs tiers, souvent hébergés hors de l'Union Européenne. Pour les organisations soumises au **RGPD**, à la directive **NIS2** ou aux réglementations sectorielles (santé, finance, défense), cette situation est problématique. Avec un LLM local, aucune donnée ne quitte votre infrastructure. Les secrets industriels, les données médicales et les informations clients restent strictement dans votre périmètre de sécurité.



Réduction des Coûts à Long Terme

Les API cloud facturent chaque token généré. Pour une entreprise traitant des millions de requêtes par mois, la facture peut atteindre plusieurs dizaines de milliers d'euros. Un investissement matériel initial (GPU, serveur) peut être amorti en quelques mois selon le volume d'utilisation. De plus, les modèles open source comme **Llama 3**, **Mistral**, **Qwen 2.5** et **DeepSeek V3** offrent des performances comparables aux modèles propriétaires pour de nombreux cas d'usage.



Latence et Disponibilité

L'inférence locale élimine la latence réseau et les temps d'attente liés aux files d'attente des fournisseurs cloud. Vous n'êtes plus dépendant de la disponibilité d'un service tiers. Pas de rate limiting, pas de pannes inattendues, pas de changements de modèle imposés par le fournisseur. Cette indépendance est cruciale pour les applications critiques en temps réel.

- 

Conformité RGPD — Les données personnelles ne quittent jamais votre infrastructure

- 

Coûts prévisibles — Investissement matériel fixe vs facturation variable à l'usage

- 

Latence réduite — Inférence directe sans transit réseau ni file d'attente

- 

Personnalisation totale — Fine-tuning, Modelfiles, templates de prompts personnalisés

- 

Indépendance technologique — Aucune dépendance à un fournisseur cloud unique

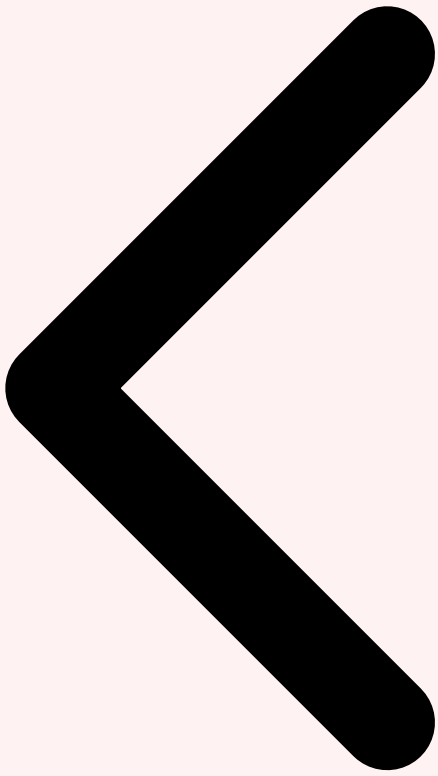
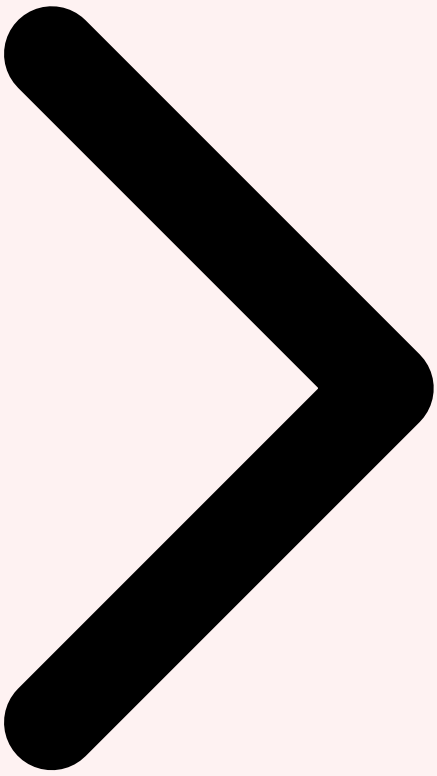
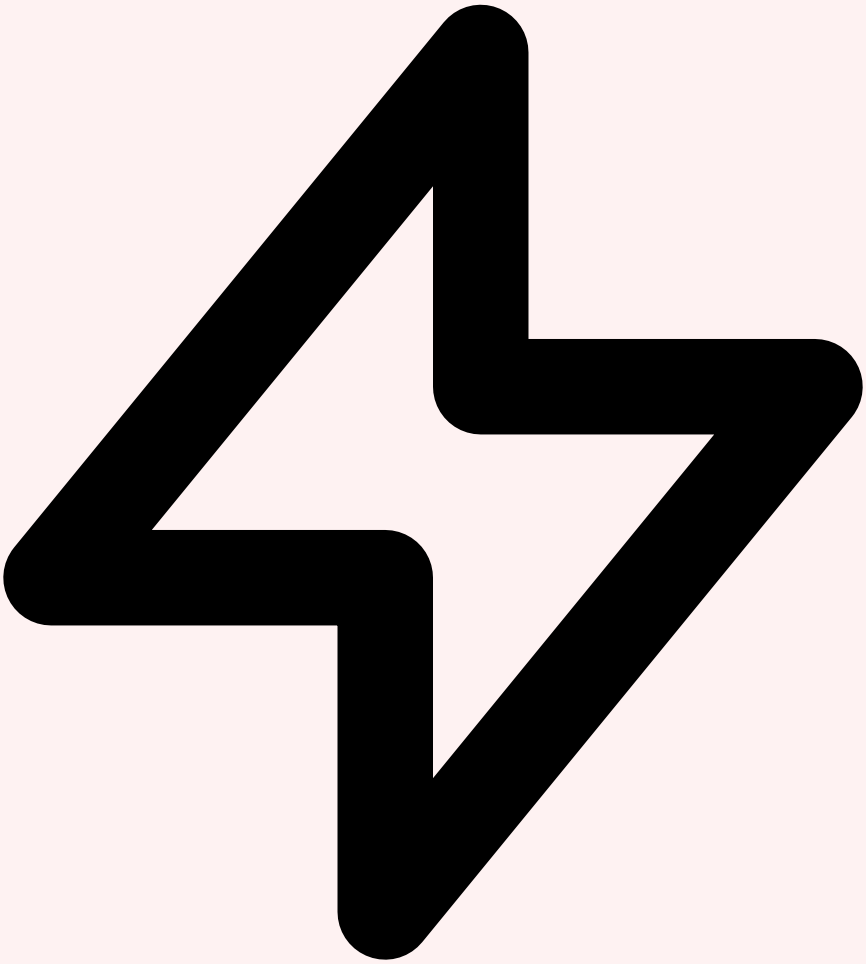


Table des Matières Pourquoi un LLM en Local Ollama



2. Ollama : La Simplicité au Service du LLM Local

Ollama est sans doute l'outil le plus populaire pour exécuter des LLM en local en 2026. Conçu pour être le « Docker des LLM », il offre une expérience utilisateur remarquablement simple. Son architecture repose sur **llama.cpp** en backend, ce qui lui permet de gérer efficacement la quantization GGUF et l'inférence sur CPU et GPU.



Architecture et Installation

Ollama fonctionne comme un **serveur d'inférence local** qui expose une API REST compatible OpenAI. L'installation est triviale sur les trois plateformes majeures :

Linux / macOS

curl -fsSL https://ollama.com/install.sh | sh

Télécharger et lancer un modèle

ollama pull llama3.3:70b

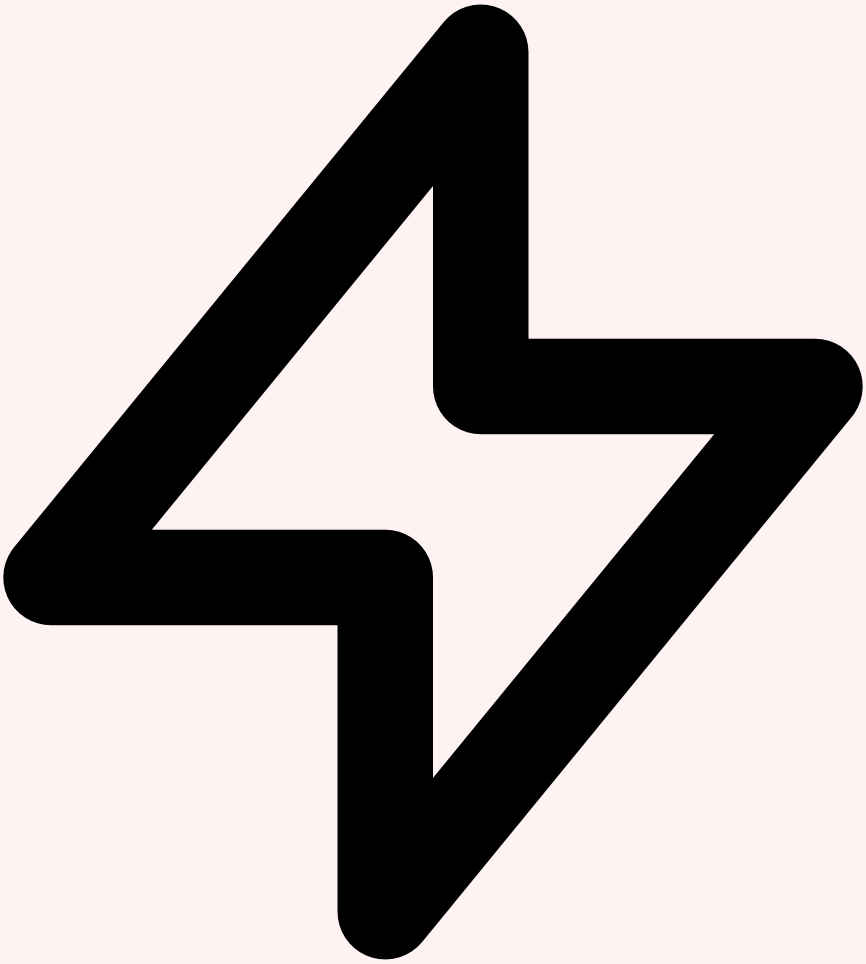
ollama run mistral:7b Pour approfondir, consultez 10 Erreurs Courantes dans.

ollama run qwen2.5:32b

Lister les modèles installés

ollama list

L'architecture interne d'Ollama s'appuie sur plusieurs composants clés : un **serveur HTTP** écrit en Go, un **moteur d'inférence** basé sur llama.cpp (C++), un **gestionnaire de modèles** avec répertoire local, et un **système de Modelfile** inspiré des Dockerfiles. Le serveur écoute par défaut sur le port 11434.



Modelfile et Personnalisation

Le système de **Modelfile** est l'une des fonctionnalités les plus puissantes d'Ollama. Inspiré de la syntaxe Dockerfile, il permet de créer des modèles personnalisés avec des paramètres spécifiques, des system prompts et des templates de conversation :

Modelfile - Assistant cybersecurity

FROM mistral:7b

PARAMETER temperature 0.3

PARAMETER num_ctx 8192

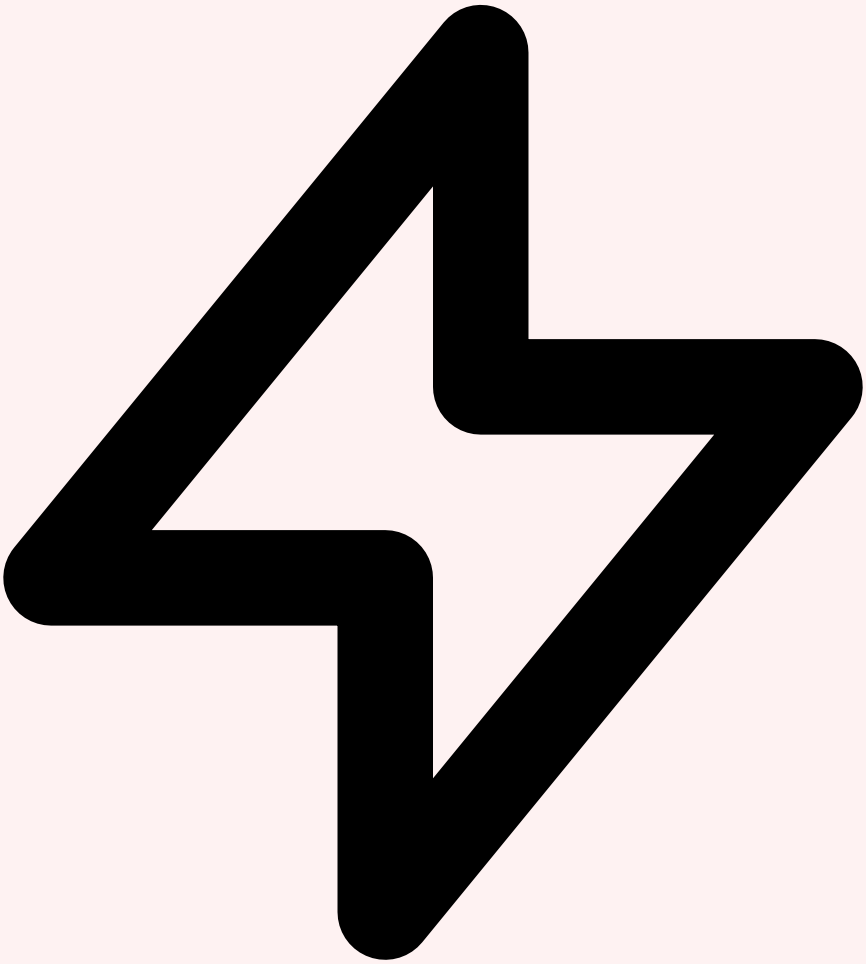
PARAMETER top_p 0.9

SYSTEM """Tu es un expert en cybersécurité spécialisé en analyse de vulnérabilités. Réponds toujours en français avec des recommandations actionables.""" Pour approfondir, consultez Détection Multimodale d'Anomalies Réseau par IA en Production.

Créer et utiliser le modèle

ollama create cyber-assistant -f Modelfile

ollama run cyber-assistant



API REST et Écosystème

Ollama expose une **API REST compatible OpenAI** sur `localhost:11434`, ce qui permet de l'intégrer facilement dans n'importe quelle application. L'écosystème autour d'Ollama est riche : **Open WebUI** fournit une interface graphique web complète, **Continue.dev** permet l'intégration dans VS Code, et les bibliothèques Python/JavaScript facilitent le

développement d'applications. La compatibilité avec le format OpenAI signifie que la plupart des outils existants fonctionnent directement avec Ollama en changeant simplement l'URL de base.

- 

Formats supportés — GGUF natif, quantizations Q4_K_M, Q5_K_M, Q8_0, FP16

- 

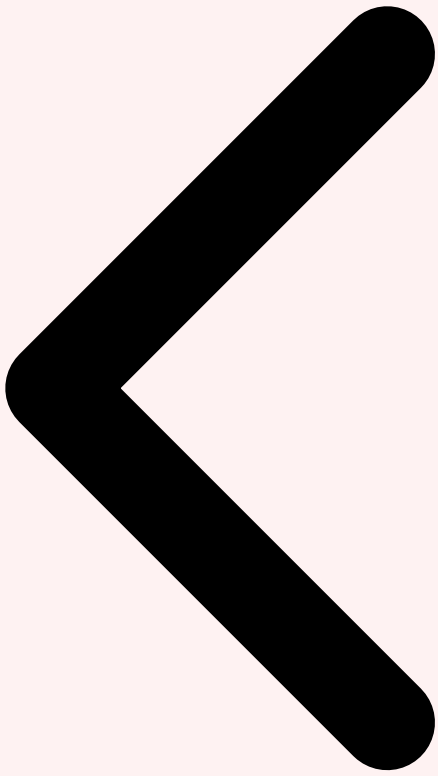
GPU — NVIDIA CUDA, AMD ROCm, Apple Metal (M1/M2/M3/M4)

- 

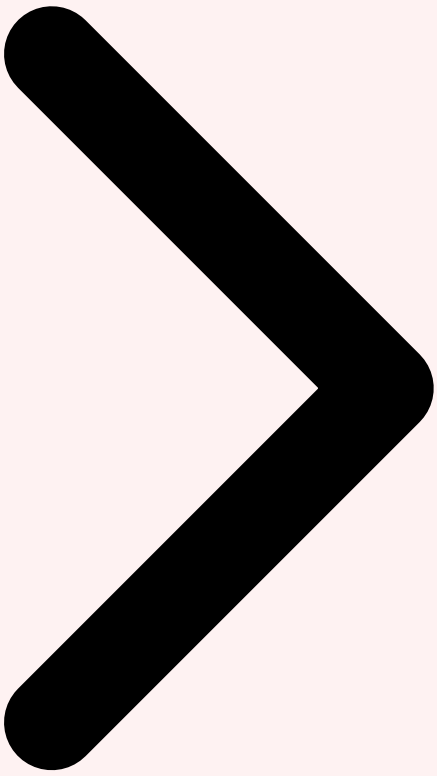
Multimodal — Support des modèles vision (LLaVA, Llama 3.2 Vision)

- 

Bibliothèque — Plus de 200 modèles prêt à l'emploi sur ollama.com/library



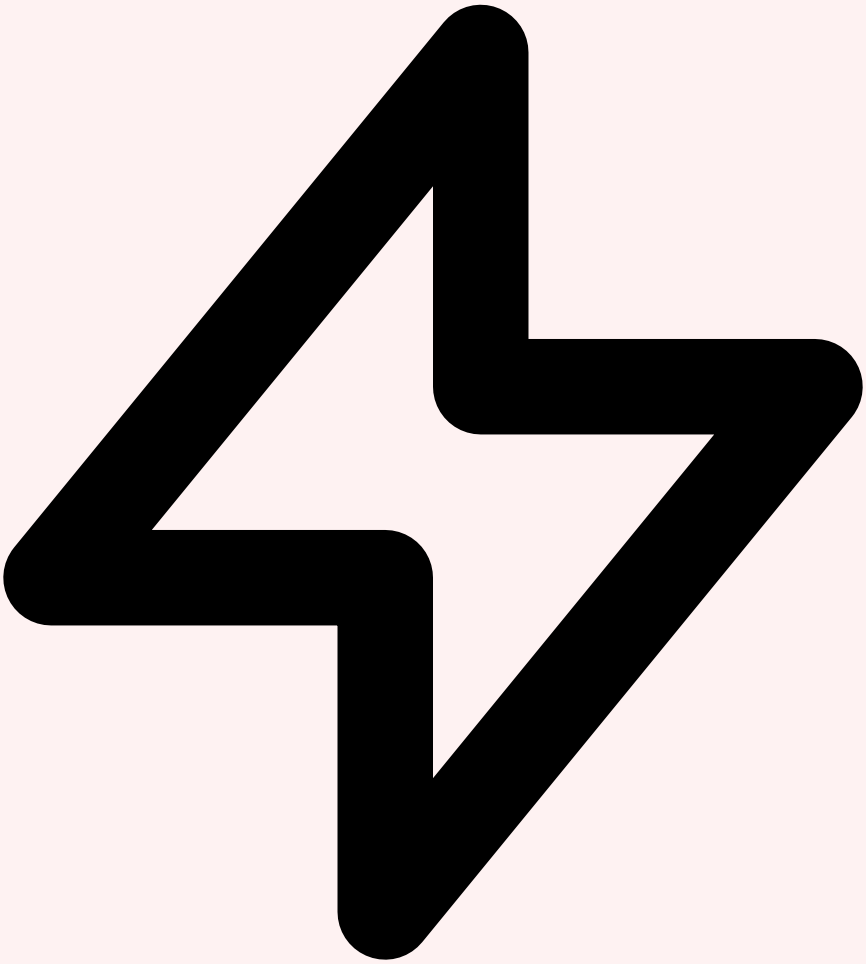
Pourquoi un LLM en Local Ollama LM Studio



Cas concret
En février 2024, une entreprise de Hong Kong a perdu 25 millions de dollars après qu'un employé a été trompé par un deepfake vidéo lors d'une visioconférence. Les attaquants avaient recréé l'apparence et la voix du directeur financier à l'aide de modèles d'IA générative, démontrant les risques concrets de cette technologie en contexte corporate.

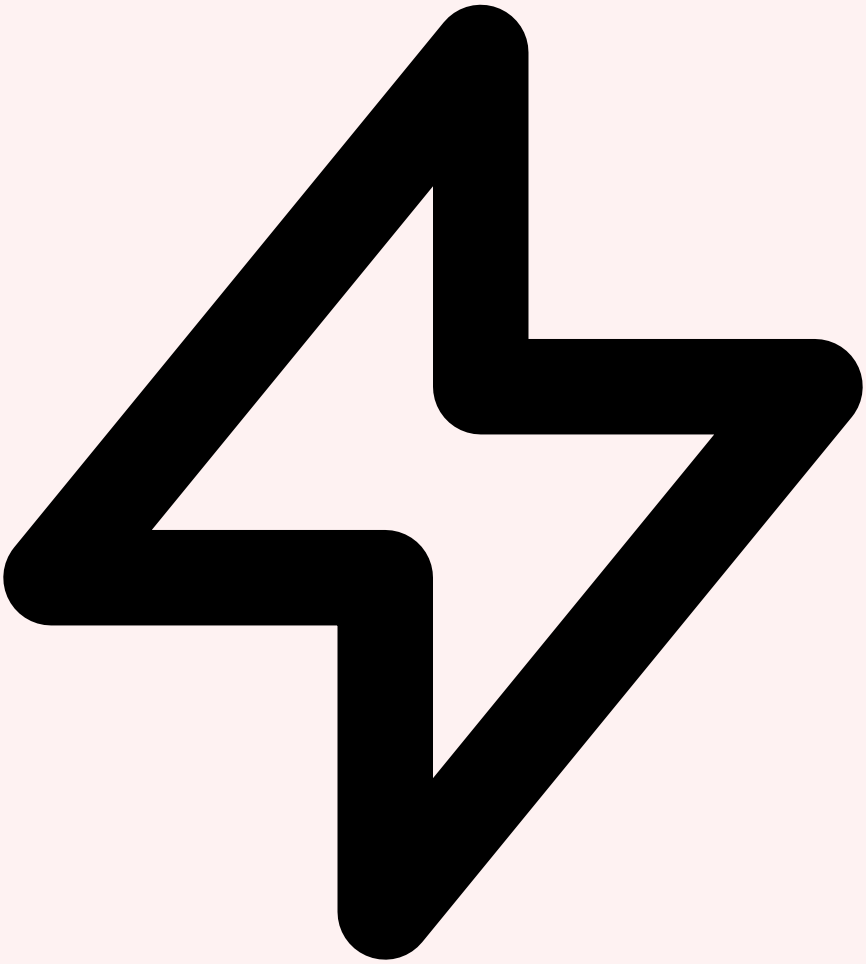
3. LM Studio : L'Interface Graphique pour les LLM

LM Studio se positionne comme la solution idéale pour les utilisateurs qui préfèrent une **interface graphique complète** plutôt qu'une ligne de commande. Développé par Element Labs, cet outil offre une expérience desktop soignée sur Windows, macOS et Linux, avec une intégration directe du catalogue **HuggingFace**.



Découverte et Téléchargement de Modèles

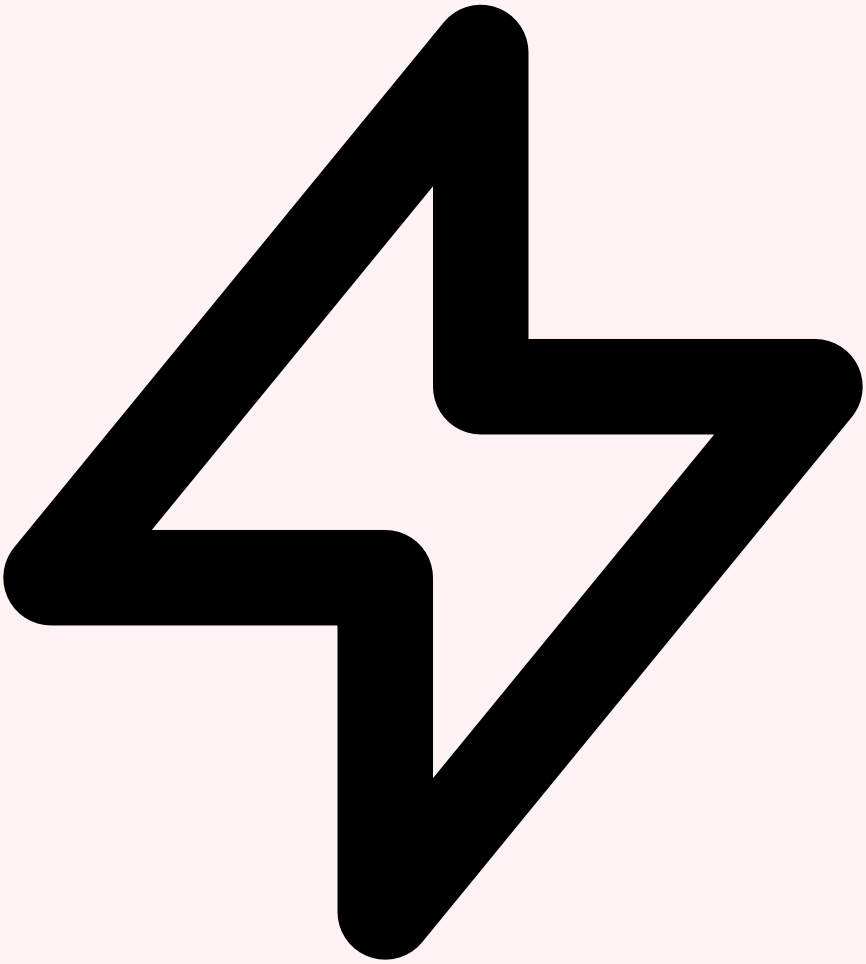
LM Studio intègre un **moteur de recherche de modèles** qui parcourt directement les dépôts HuggingFace. L'utilisateur peut filtrer par architecture (Llama, Mistral, Phi, Gemma), par taille (7B, 13B, 34B, 70B), par format de quantization (GGUF, GPTQ) et par compatibilité matérielle. Un système de recommandation indique automatiquement si le modèle choisi peut fonctionner sur votre machine en fonction de la VRAM et de la RAM disponibles.



Interface de Chat et Paramétrage

L'interface de chat de LM Studio est l'une des plus abouties du marché. Elle propose un panneau de configuration latéral avec tous les hyperparamètres d'inférence : **temperature**, **top_p**, **top_k**, **repeat_penalty**, **max_tokens**, et bien d'autres. Un mode **multi-modèle** permet de comparer les réponses de différents modèles côte à côte, ce qui est particulièrement utile pour le benchmarking qualitatif.

Le **profiling intégré** affiche en temps réel les métriques de performance : tokens par seconde (t/s), utilisation VRAM, utilisation CPU/GPU, et temps de première réponse (Time to First Token - TTFT). Ces informations sont précieuses pour optimiser la configuration et choisir le bon niveau de quantization.



Serveur API Local

LM Studio embarque un **serveur API local** compatible avec le format OpenAI. En un clic, vous pouvez démarrer un serveur HTTP qui expose les endpoints `/v1/chat/completions` et `/v1/completions`. Cette fonctionnalité transforme LM Studio en véritable backend d'inférence pour vos applications. Le serveur supporte le streaming SSE (Server-Sent Events), l'embeddings, et depuis la version 0.3, le function calling.

- 

Avantage clé — Interface intuitive idéale pour l'exploration et le prototypage rapide

- 

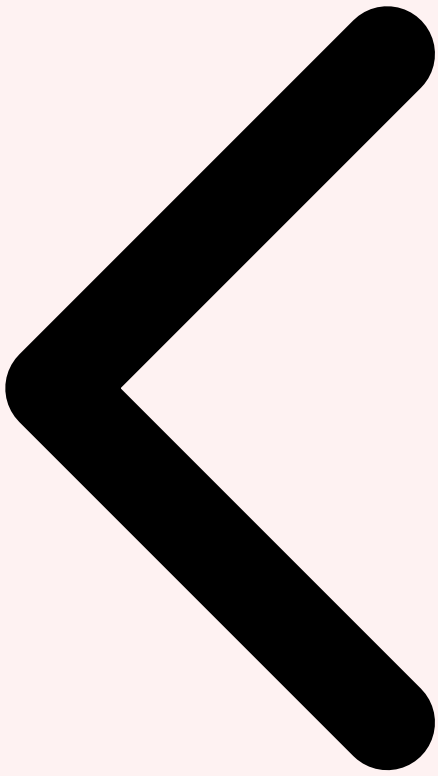
Catalogue HuggingFace — Accès direct à des milliers de modèles GGUF

- 

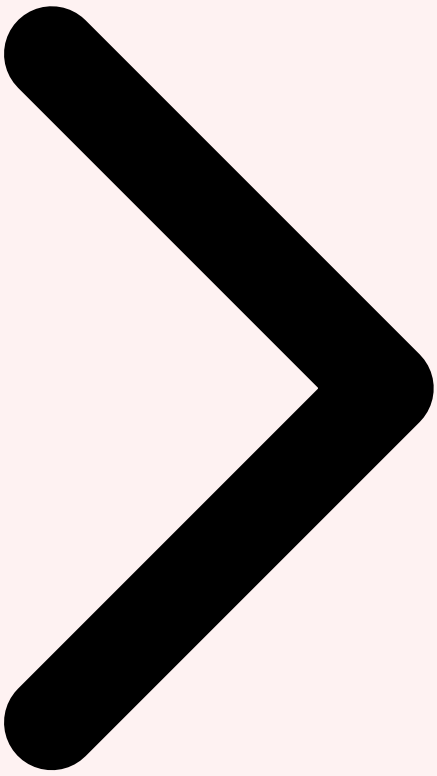
Profiling temps réel — Métriques de performance visibles pendant l'inférence

- 

Multi-plateforme — Windows, macOS (Apple Silicon natif), Linux



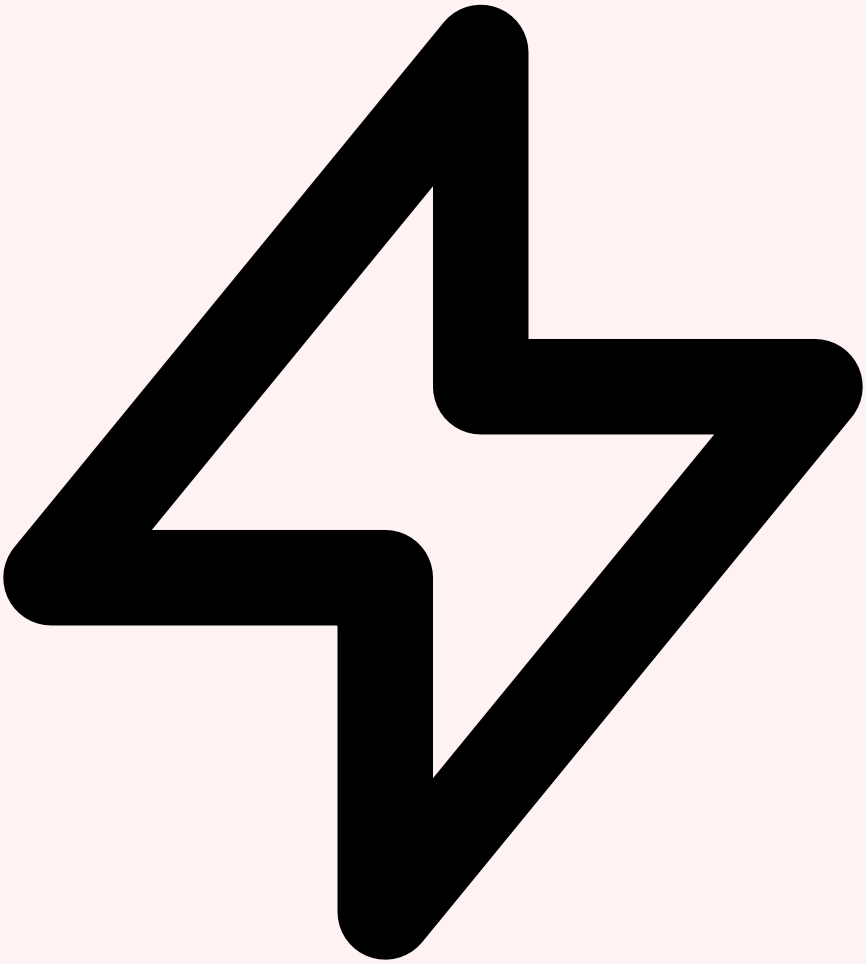
Ollama LM Studio vLLM



Comment garantir que vos modèles de machine learning ne deviennent pas des vecteurs d'attaque ?

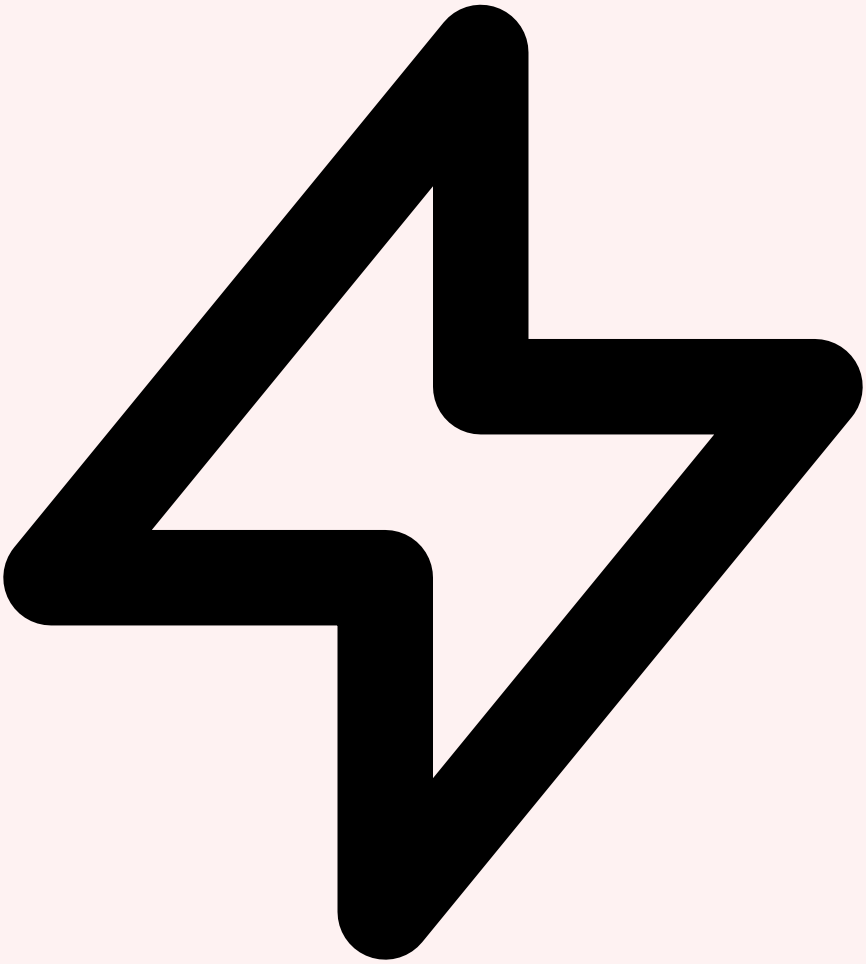
4. vLLM : Le Moteur d'Inférence Haute Performance

vLLM est un moteur d'inférence haute performance développé initialement par l'équipe de recherche de UC Berkeley. Contrairement à Ollama et LM Studio, vLLM est conçu dès le départ pour les **déploiements en production** nécessitant un débit élevé et une gestion optimale de la concurrence.



PagedAttention : L'Innovation Clé

La principale innovation de vLLM est le mécanisme de **PagedAttention**. Inspiré de la gestion de la mémoire virtuelle des systèmes d'exploitation, PagedAttention découpe le cache KV (Key-Value) en blocs de taille fixe et les alloue à la demande. Cette approche réduit le gaspillage mémoire de 60 à 80% par rapport aux méthodes traditionnelles d'allocation contiguë. En pratique, cela signifie que vLLM peut servir **2 à 4 fois plus de requêtes simultanées** qu'un moteur classique avec la même quantité de VRAM. Pour approfondir, consultez Playbooks de Réponse aux Incidents IA : Modèles et Automatisation.



Continuous Batching et Tensor Parallelism

vLLM implémente le **continuous batching** (ou iteration-level scheduling), une technique qui permet d'ajouter de nouvelles requêtes au batch en cours sans attendre que toutes les requêtes précédentes soient terminées. Le moteur supporte également le **tensor parallelism** pour distribuer un modèle sur plusieurs GPU, ce qui est indispensable pour les modèles de grande taille (70B+). La configuration est simple :

Installation

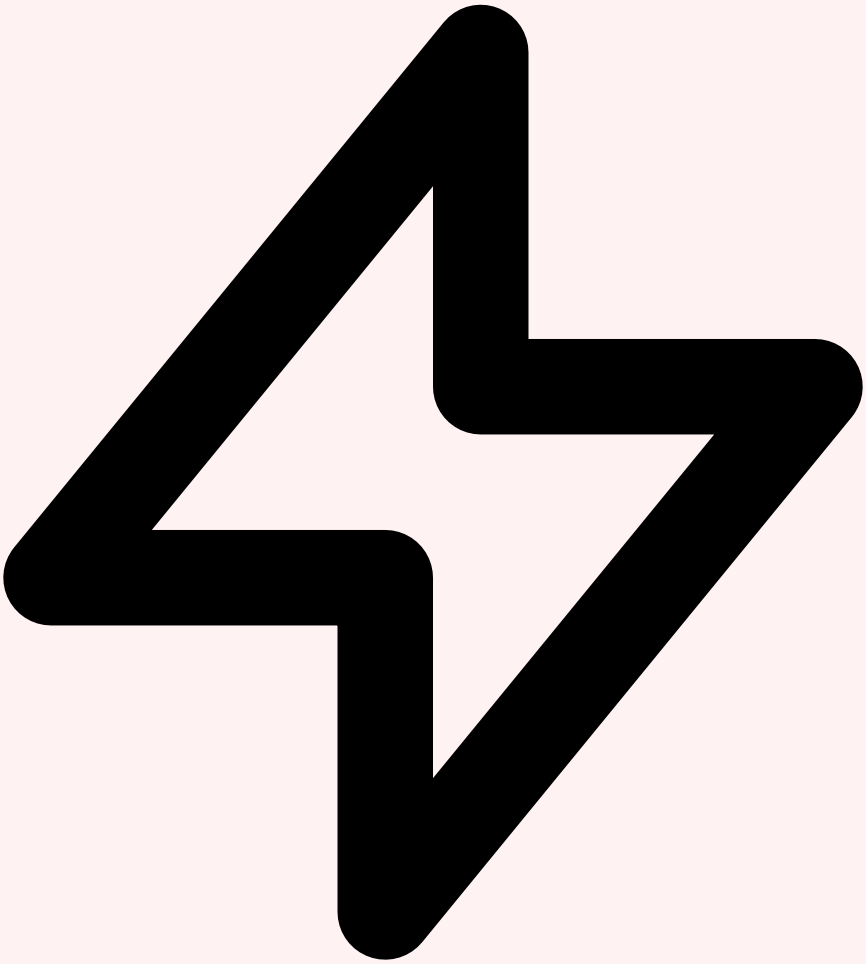
pip install vllm

Lancer un serveur compatible OpenAI

vllm serve meta-llama/Llama-3.3-70B-Instruct --tensor-parallel-size 4 --gpu-memory-utilization 0.90

Avec quantization AWQ

vllm serve TheBloke/Mistral-7B-Instruct-v0.3-AWQ --quantization awq --max-model-len 32768



Fonctionnalités Production

vLLM offre un ensemble complet de fonctionnalités orientées production. Le **speculative decoding** utilise un petit modèle draft pour accélérer l'inférence du modèle principal. Le **prefix caching** met en cache les préfixes de prompts fréquents pour éviter les recalculs. Le

support natif de **LoRA** permet de charger dynamiquement des adaptateurs fine-tunés sans redémarrer le serveur. Enfin, les métriques **Prometheus** intégrées facilitent le monitoring en production.

- 

PagedAttention — Gestion optimale du cache KV, réduction de 60-80% du gaspillage mémoire

- 

Tensor Parallelism — Distribution multi-GPU pour les modèles de grande taille

- 

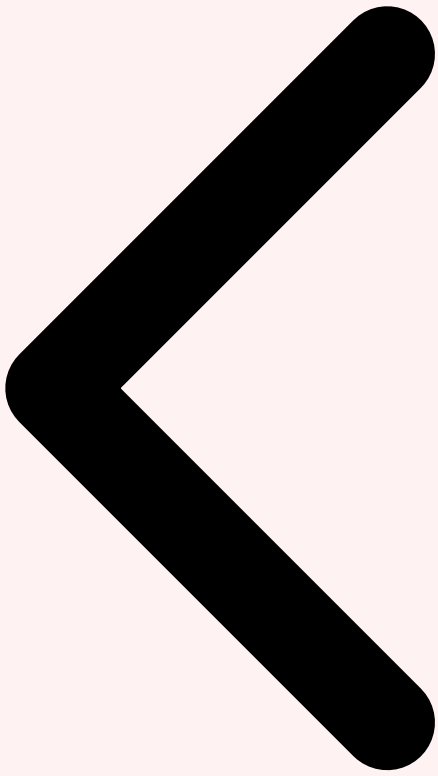
Continuous Batching — Ajout dynamique de requêtes au batch en cours d'exécution

- 

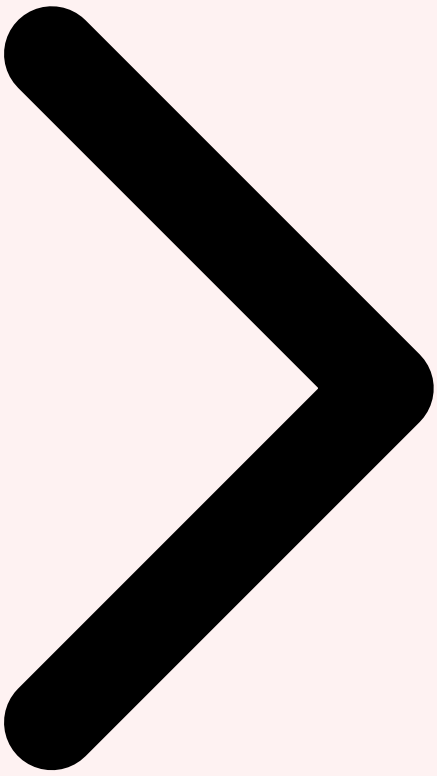
Quantization — AWQ, GPTQ, SqueezeLLM, FP8 pour optimiser l'empreinte mémoire

- 

Monitoring — Métriques Prometheus, logs structurés, intégration Grafana



LM Studio vLLM Comparatif Détaillé



5. Comparatif Détaillé : Ollama vs LM Studio vs vLLM

Pour choisir le bon outil, comparer ces trois solutions sur des critères objectifs. Le tableau ci-dessous synthétise les différences majeures en termes de facilité d'utilisation, performance, écosystème et cas d'usage cibles.

Critère	Ollama	LM Studio	vLLM
Interface	CLI + API REST	GUI Desktop + API	CLI + API REST
Facilité d'installation	Très facile (1 commande)	Très facile (installer .exe/.dmg)	Moyen (pip + CUDA)
Formats de modèles	GGUF	GGUF, GPTQ	HF, AWQ, GPTQ, FP8
Backend	llama.cpp (C++)	llama.cpp (C++)	PyTorch + CUDA kernels
CPU uniquement	Oui (performant)	Oui (performant)	Limité (GPU recommandé)
Multi-GPU	Basique	Non	Tensor Parallelism natif
Concurrent batching	Non (séquentiel)	Non (séquentiel)	Oui (continuous batching)
Throughput (requêtes/s)	Faible-moyen	Faible-moyen	Élevé (2-4x supérieur)
Apple Silicon	Excellent (Metal)	Excellent (Metal)	Non supporté
Cas d'usage principal	Développement, prototypage	Exploration, test	Production, haute charge
Licence	MIT (open source)	Propriétaire (gratuit)	Apache 2.0 (open source)

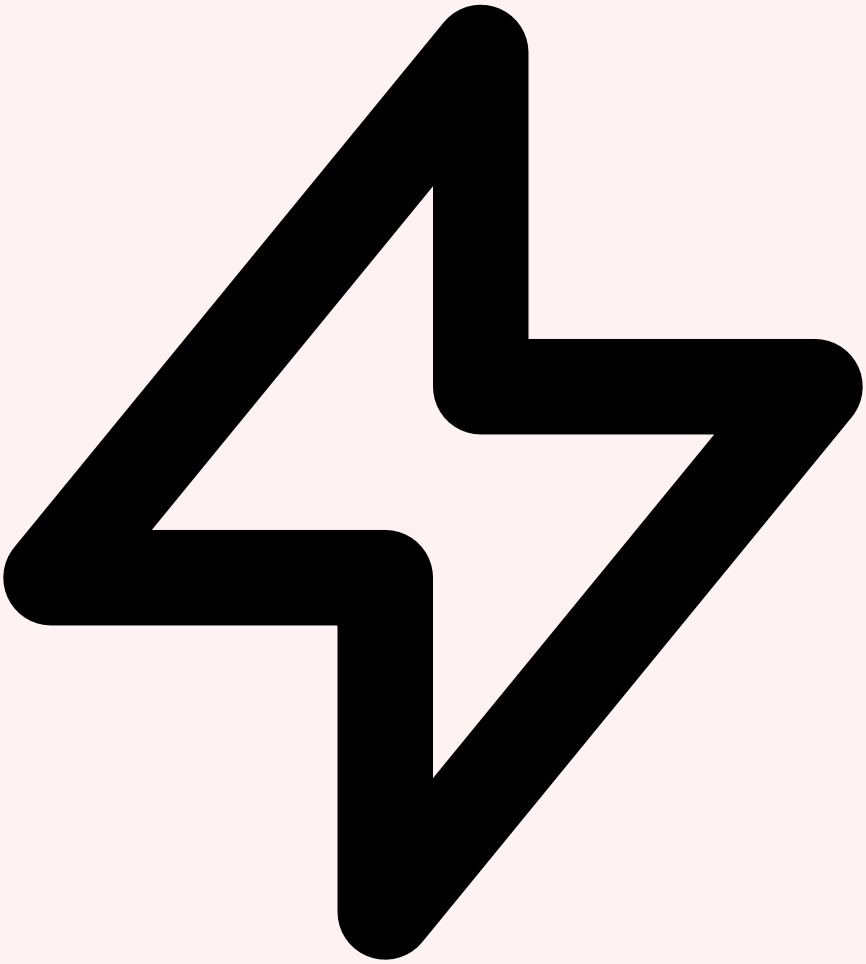


Diagramme d'Architecture Comparée

Le diagramme suivant illustre les différences architecturales fondamentales entre les trois outils. Ollama et LM Studio partagent le même moteur llama.cpp mais diffèrent dans leur couche d'interface, tandis que vLLM adopte une approche radicalement différente basée sur PyTorch et des kernels CUDA optimisés.

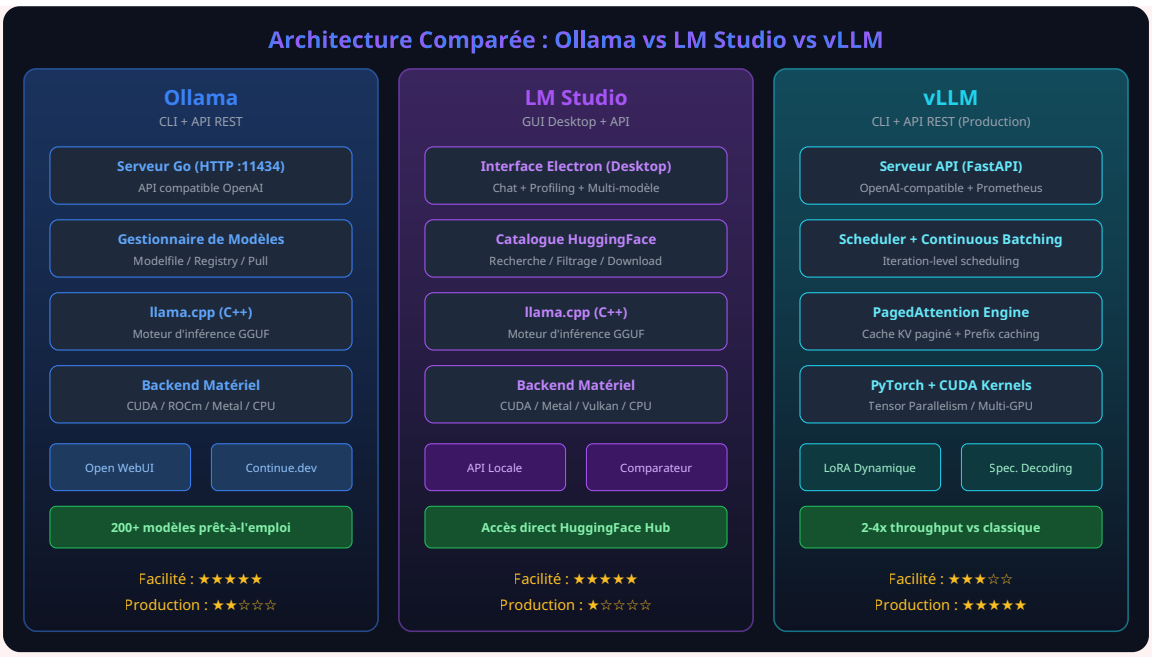
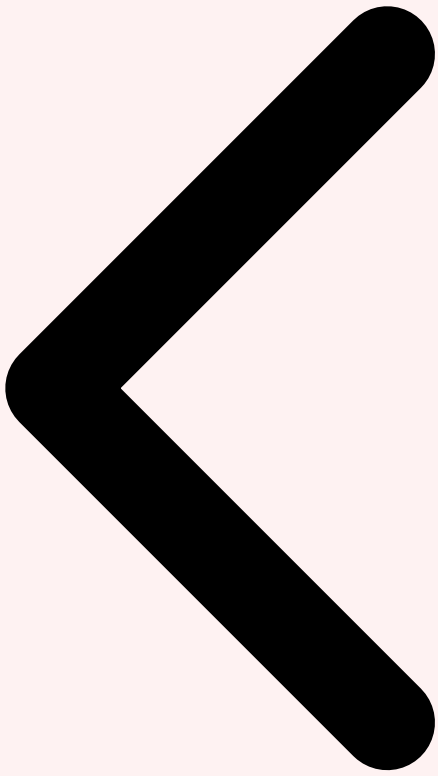
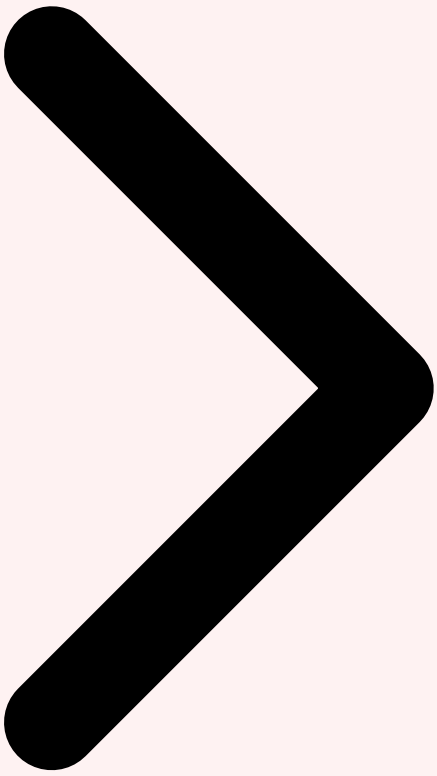


Fig. 1 — Architecture comparée des trois moteurs d'inférence LLM local Pour approfondir, consultez Shadow Agents IA : Identification, Gouvernance et Remédiation.

On observe que **Ollama et LM Studio partagent le même moteur llama.cpp**, ce qui explique des performances brutes similaires pour un seul utilisateur. La différence principale réside dans l'expérience utilisateur : CLI élégante pour Ollama, GUI pour LM Studio. **vLLM**, en revanche, adopte une architecture fondamentalement différente avec PyTorch et des optimisations CUDA de bas niveau, ce qui lui confère un avantage décisif en environnement multi-utilisateurs et haute charge.



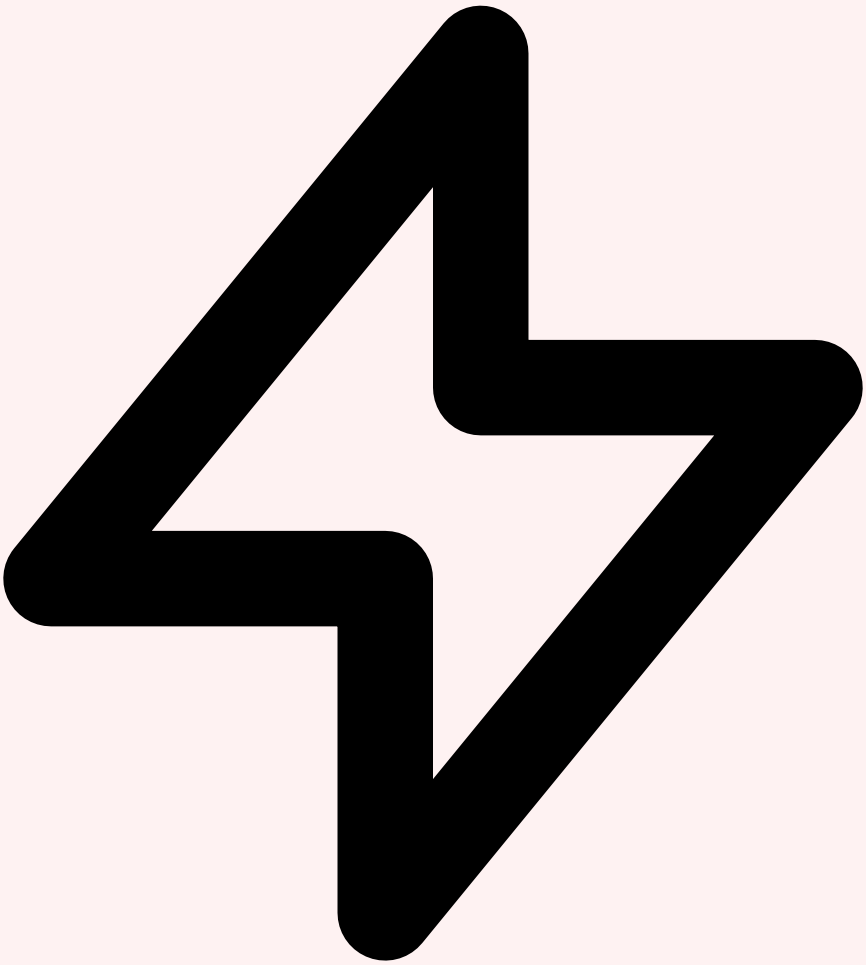
vLLM Comparatif Détaillé Configuration Matérielle



6. Configuration Matérielle : GPU, RAM et VRAM

Le choix du matériel est déterminant pour les performances de votre LLM local. La règle fondamentale est simple : **plus le modèle est grand, plus il faut de mémoire**. Un modèle 7B quantizé en Q4 occupe environ 4 Go, tandis qu'un 70B en Q4 nécessite environ 40 Go. Voici les configurations recommandées par taille de modèle.

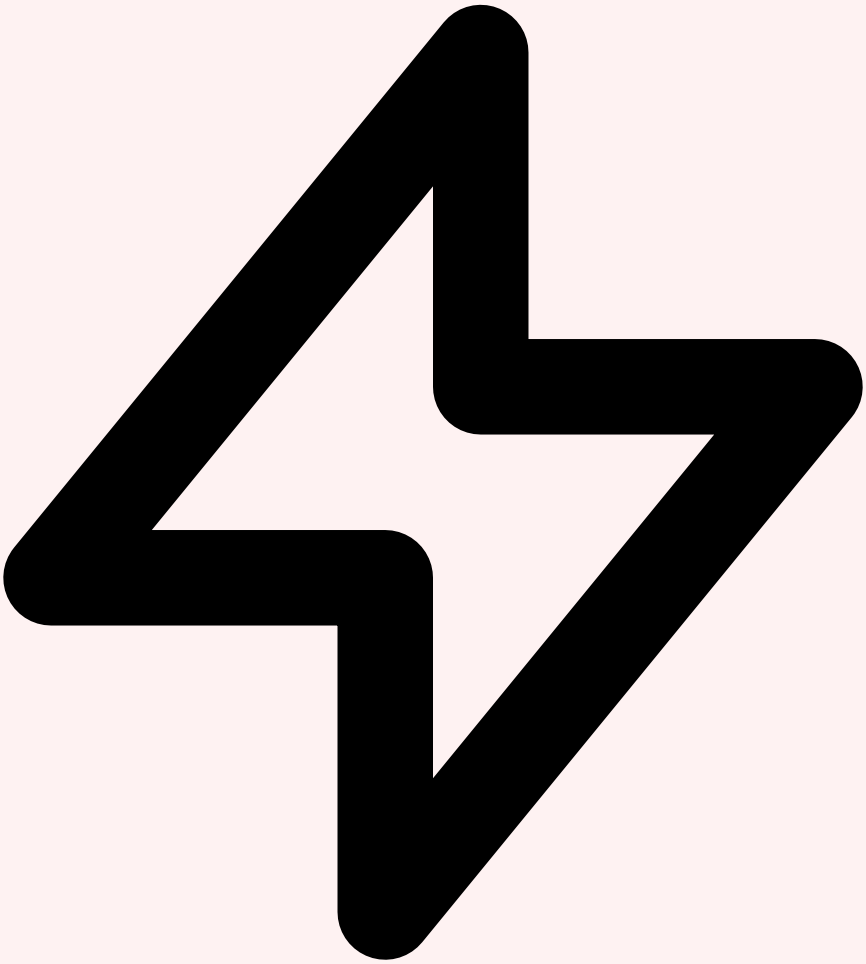
Taille modèle	VRAM (Q4)	RAM min.	GPU recommandé	Alternative
1-3B	2-3 Go	8 Go	Intégré / GTX 1660	CPU uniquement
7-8B	4-6 Go	16 Go	RTX 3060 12Go / RTX 4060 Ti	Mac M1/M2 16Go
13-14B	8-10 Go	32 Go	RTX 4070 Ti 12Go	Mac M2 Pro 32Go
32-34B	20-24 Go	48 Go	RTX 4090 24Go / RTX A5000	Mac M3 Max 48Go
70B	40-48 Go	64 Go	2x RTX 4090 / A100 80Go	Mac M3 Ultra 128Go
120-405B	80-240 Go	128+ Go	4-8x A100 / H100	Mac M4 Ultra 256Go (partiel)



NVIDIA vs AMD vs Apple Silicon

NVIDIA reste la référence pour l'inférence LLM grâce à l'écosystème CUDA mature, au support de tous les frameworks (vLLM, TensorRT-LLM, llama.cpp) et aux optimisations de bas niveau (FlashAttention, FP8). **AMD** progresse rapidement avec ROCm et les RX 7900 XTX

(24 Go VRAM), mais le support logiciel reste en retrait. **Apple Silicon** offre un excellent rapport qualité/prix pour l'utilisation locale avec sa mémoire unifiée (jusqu'à 256 Go sur M4 Ultra), et fonctionne parfaitement avec Ollama et LM Studio via Metal.



Benchmark de Performance

Le graphique ci-dessous présente les performances d'inférence (tokens par seconde) mesurées sur différentes configurations matérielles pour chaque outil, en utilisant Mistral 7B Q4_K_M comme modèle de référence.

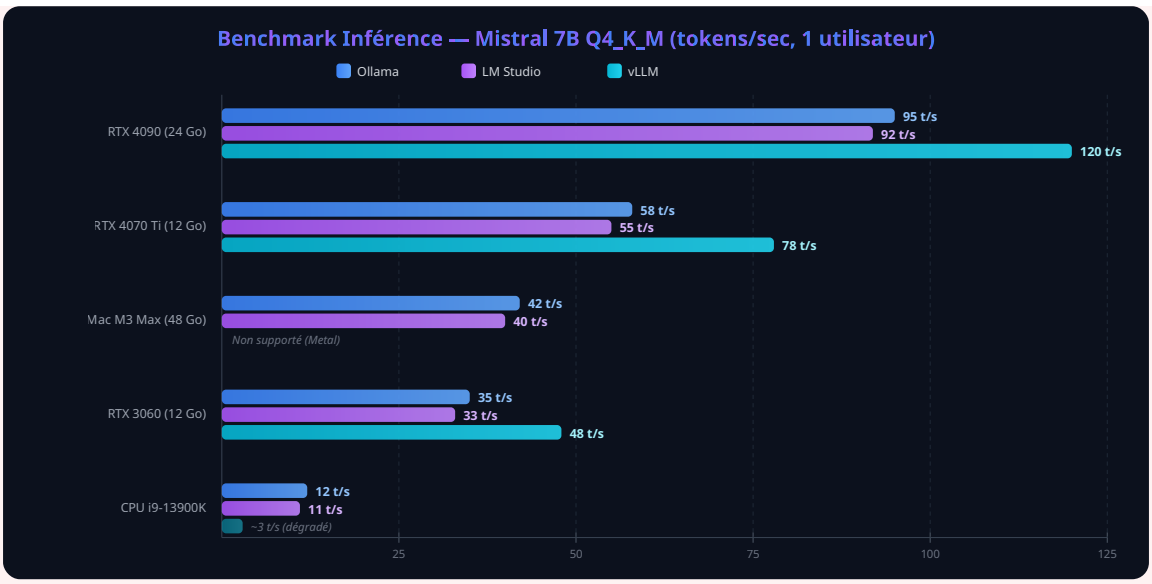
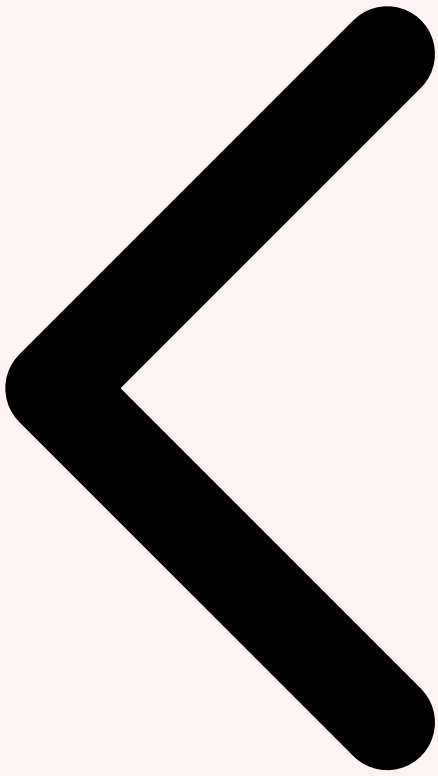
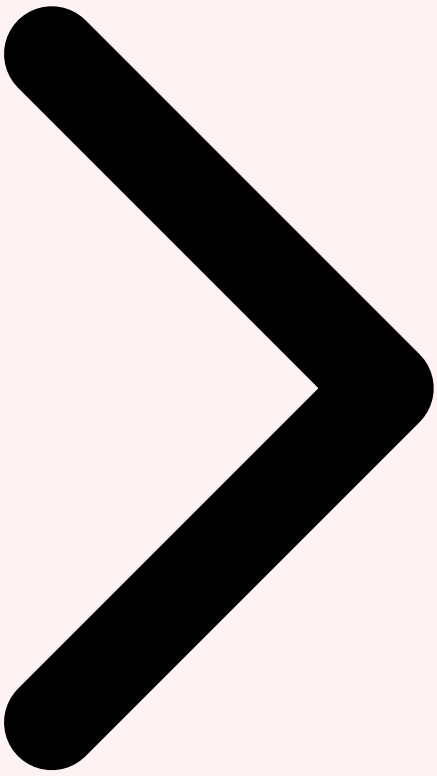


Fig. 2 — Benchmark tokens/sec sur Mistral 7B Q4_K_M (1 utilisateur, génération 512 tokens)

Ces benchmarks confirment plusieurs tendances. Sur GPU NVIDIA, **vLLM surpasse systématiquement** Ollama et LM Studio grâce à ses optimisations CUDA. L'écart se creuse davantage en mode multi-utilisateurs où le continuous batching de vLLM permet de maintenir un débit élevé. Sur **Apple Silicon**, Ollama et LM Studio offrent d'excellentes performances grâce à Metal, tandis que vLLM n'est pas compatible. Pour l'utilisation **CPU uniquement**, Ollama et LM Studio restent les meilleurs choix grâce aux optimisations AVX2/AVX-512 de llama.cpp.

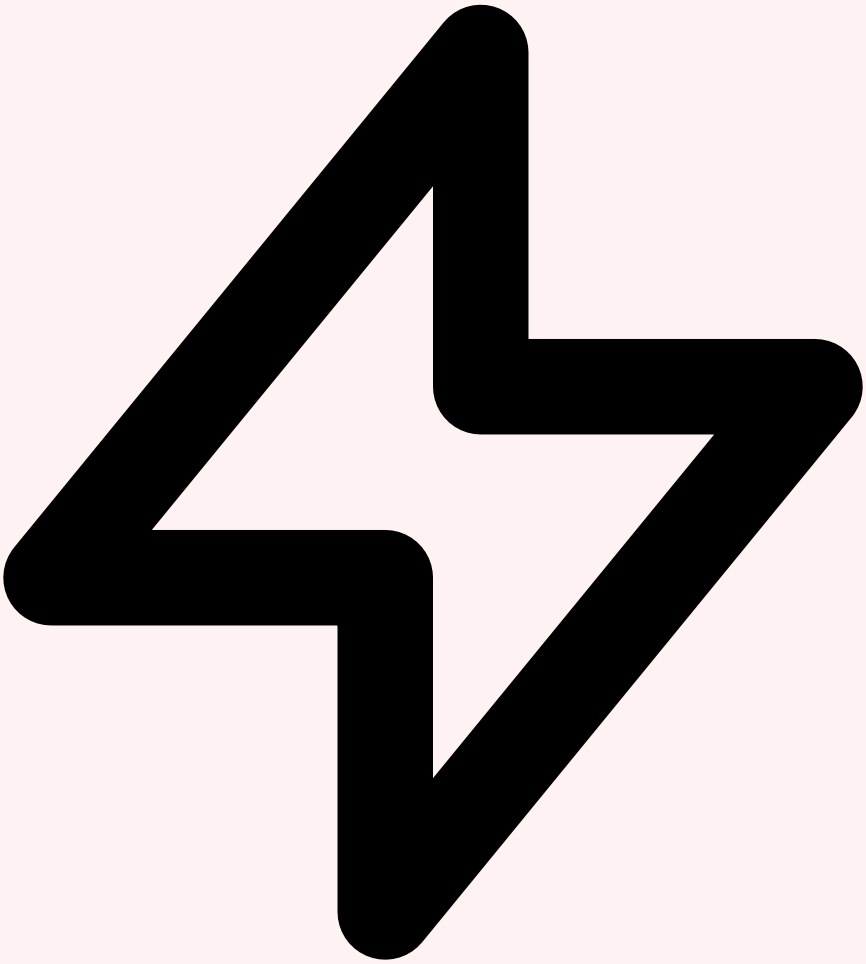


Comparatif Détaillé Configuration Matérielle Guide de Choix



7. Guide de Choix et Cas d'Usage

Le choix entre Ollama, LM Studio et vLLM dépend fondamentalement de votre **profil d'utilisateur**, de votre **infrastructure** et de vos **objectifs**. Voici un guide détaillé par scénario.



Choisissez Ollama si...

- 

Vous êtes **développeur** et préférez travailler en ligne de commande

- 

Vous voulez **intégrer un LLM dans une application** via API REST compatible OpenAI

- 

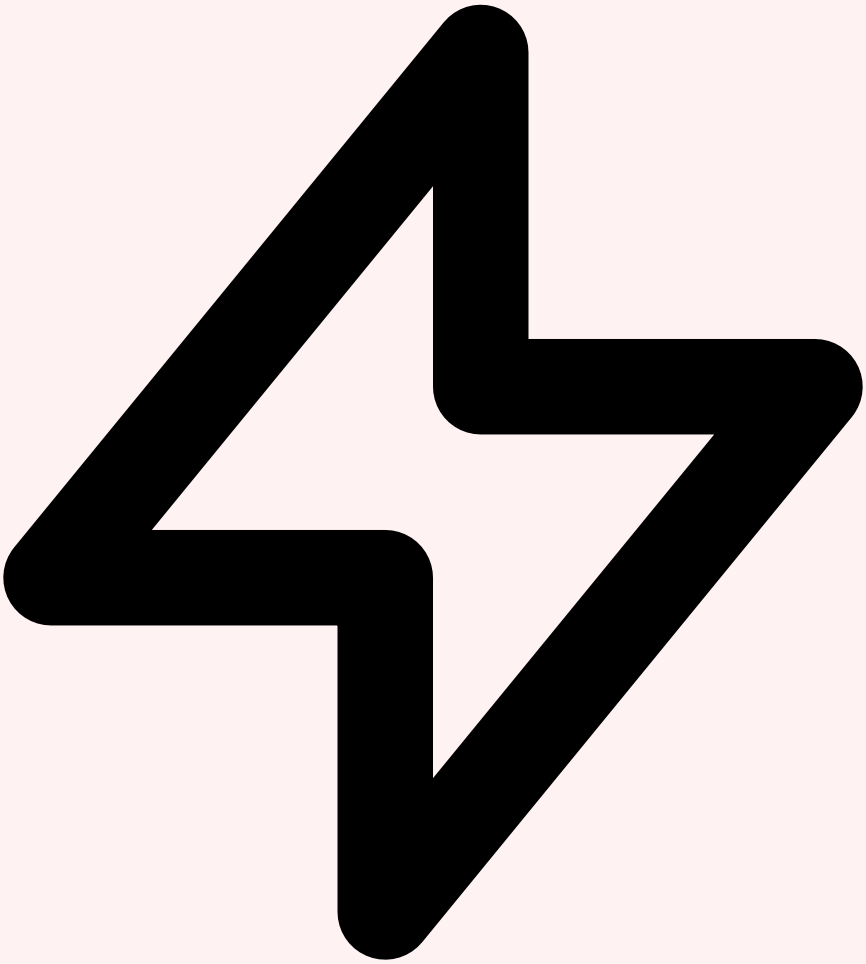
Vous avez besoin de **Modelfiles personnalisés** pour différents cas d'usage

- 

Vous utilisez un **Mac Apple Silicon** ou un PC avec GPU NVIDIA

- 

Vous cherchez un outil **open source** avec une communauté très active



Choisissez LM Studio si...

- 

Vous préférez une **interface graphique** intuitive et soignée

- 

Vous voulez **explorer et comparer** différents modèles facilement

- 

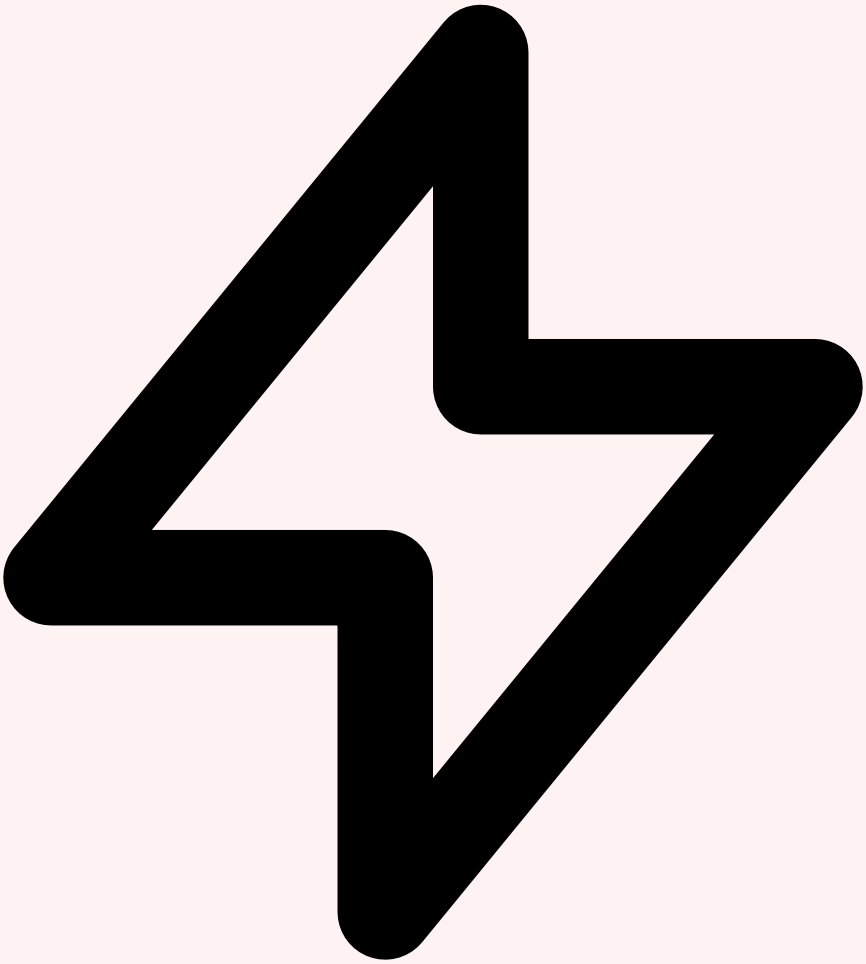
Vous souhaitez **télécharger directement** depuis HuggingFace Hub

- 

Vous êtes **débutant** et voulez une prise en main immédiate sans ligne de commande

- 

Le **profiling en temps réel** des performances vous intéresse pour optimiser vos choix



Choisissez vLLM si...

- 

Vous déployez un LLM **en production** avec de multiples utilisateurs simultanés

- 

Vous avez besoin de **tensor parallelism** sur plusieurs GPU NVIDIA

- 

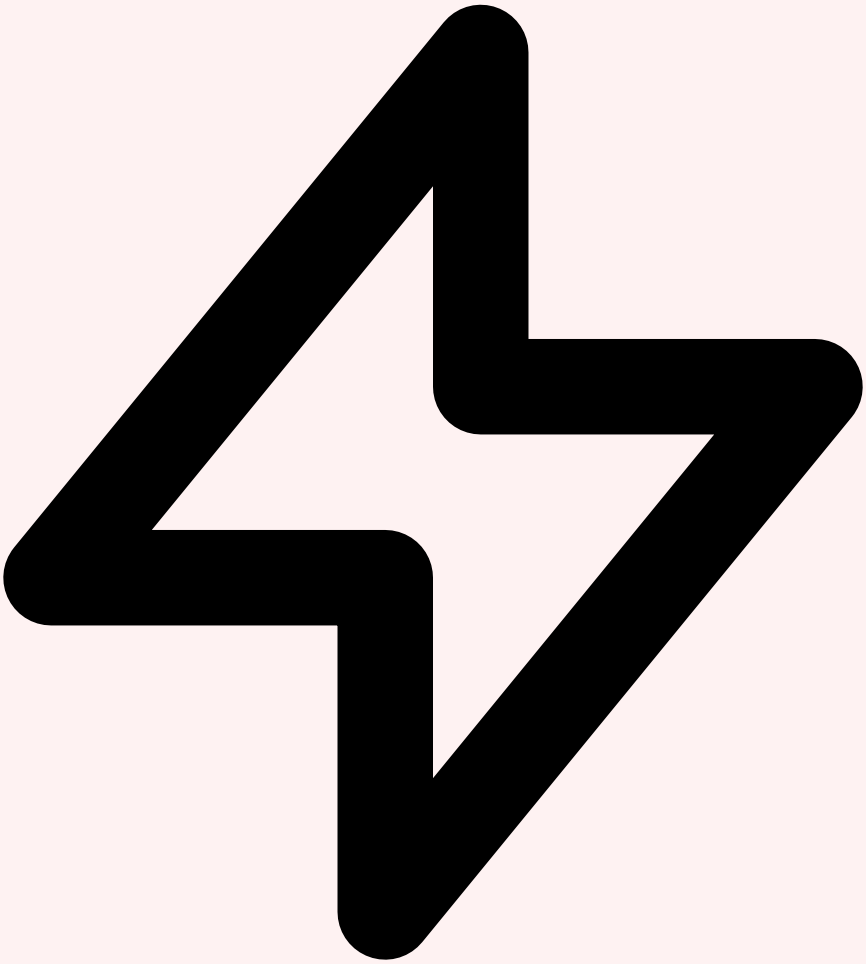
Le **throughput maximal** et la gestion de la concurrence sont prioritaires

- 

Vous avez besoin de **monitoring Prometheus/Grafana** et de LoRA dynamique

- 

Vous travaillez avec des **modèles HuggingFace au format natif** (AWQ, GPTQ, FP8)



L'Approche Combinée : La Meilleure Stratégie

En pratique, de nombreuses organisations adoptent une **approche combinée**. Le workflow typique consiste à utiliser **LM Studio** pour l'exploration et le test de nouveaux modèles, **Ollama** pour le développement quotidien et le prototypage avec son API et son écosystème riche, puis **vLLM** pour le déploiement en production avec ses optimisations de performances. Cette stratégie en trois phases permet de bénéficier des forces de chaque outil au moment le plus opportun.

L'écosystème des LLM locaux évolue rapidement. De nouveaux outils comme **llama-server** (intégré à llama.cpp), **LocalAI**, et **Jan.ai** enrichissent le paysage. Le dénominateur commun reste la **compatibilité API OpenAI**, qui facilite la migration entre les différentes solutions. Quelle que soit votre choix initial, vous conservez la flexibilité de changer d'outil sans réécrire votre code applicatif.

Résumé : Quel outil pour quel profil ?

Ollama — Le couteau suisse du développeur. Simple, rapide, extensible. Idéal pour 80% des cas d'usage.

- 

LM Studio — La porte d'entrée visuelle. Parfait pour l'exploration et la comparaison de modèles.

•

vLLM — Le champion de la production. Performances maximales, scaling multi-GPU, monitoring avancé.

Besoin d'un accompagnement expert ?

Nos consultants en cybersécurité et IA vous accompagnent dans vos projets. Devis personnalisé sous 24h.

Références et ressources externes

- vLLM — Moteur d'inférence LLM haute performance
- llama.cpp — Inférence LLM optimisée en C/C++
- MLflow — Plateforme open source de gestion du cycle de vie ML
- Kubernetes Docs — Documentation officielle Kubernetes
- HuggingFace Docs — Documentation de référence pour les modèles de ML

Comment choisir entre Ollama, LM Studio et vLLM pour deployer un LLM en local ?

Le choix depend du cas d'usage et du niveau d'expertise. Ollama est ideal pour les developpeurs souhaitant integrer rapidement un LLM via une API REST simple, avec une installation en une commande et un catalogue de modeles pre-optimises. LM Studio offre une interface graphique intuitive parfaite pour l'experimentation et le prototypage sans competences DevOps. vLLM est concu pour la production a haute performance, avec le PagedAttention pour un throughput optimal, le batching continu et le support multi-GPU, mais necessite une expertise technique plus avancee.

Pour approfondir ce sujet, consultez notre outil open-source llm-vulnerability-scanner qui facilite l'analyse des vulnérabilités des LLM.

Evaluation des risques et contre-mesures

Quels sont les prerequis materiels pour executer un LLM localement ?

Les prerequis dependent de la taille du modele. Pour un modele 7B parametres en quantification Q4, il faut minimum 8 Go de RAM GPU (ou 16 Go de RAM CPU avec des performances reduites). Un modele 13B necessite 16 Go de VRAM, et un 70B demande 40 Go minimum ou une configuration multi-GPU. Les GPU NVIDIA avec support CUDA sont recommandes pour les meilleures performances, bien que les puces Apple Silicon (M1/M2/M3) offrent un bon rapport performance-prix grace a leur memoire unifiee partagee entre CPU et GPU.

Pourquoi deployer un LLM en local plutot qu'utiliser une API cloud ?

Le deploiement local offre plusieurs avantages decisifs : la confidentialite totale des donnees qui ne quittent jamais l'infrastructure, l'absence de couts recurrents par token qui peuvent devenir prohibitifs a grande echelle, une latence reduite pour les applications temps reel, et l'independance vis-a-vis des fournisseurs cloud. C'est particulierement pertinent pour les secteurs reglementes (sante, finance, defense) soumis a des contraintes de souverainete des donnees, et pour les entreprises traitant des donnees sensibles comme du code source proprietaire ou des documents confidentiels.

Sources et références : ArXiv IA · Hugging Face Papers

Conclusion

Cet article a couvert les aspects essentiels de Table des Matières, 1. Pourquoi Exécuter un LLM en Local ?, 2. Ollama : La Simplicité au Service du LLM Local. La mise en pratique de ces recommandations permet de renforcer significativement la posture de securite de votre organisation.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr