AYI NEDJIMI Consultants

Deployer des LLM en Production : GPU et Optimisation

Catégorie : Intelligence Artificielle Lecture : 27 min Publié le : 13/02/2026 Auteur : Ayi NEDJIMI

Guide complet pour déployer des LLM en production : architecture de serving, GPU selection, scaling horizontal et vertical, optimisation d'inférence.

Table des Matières

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

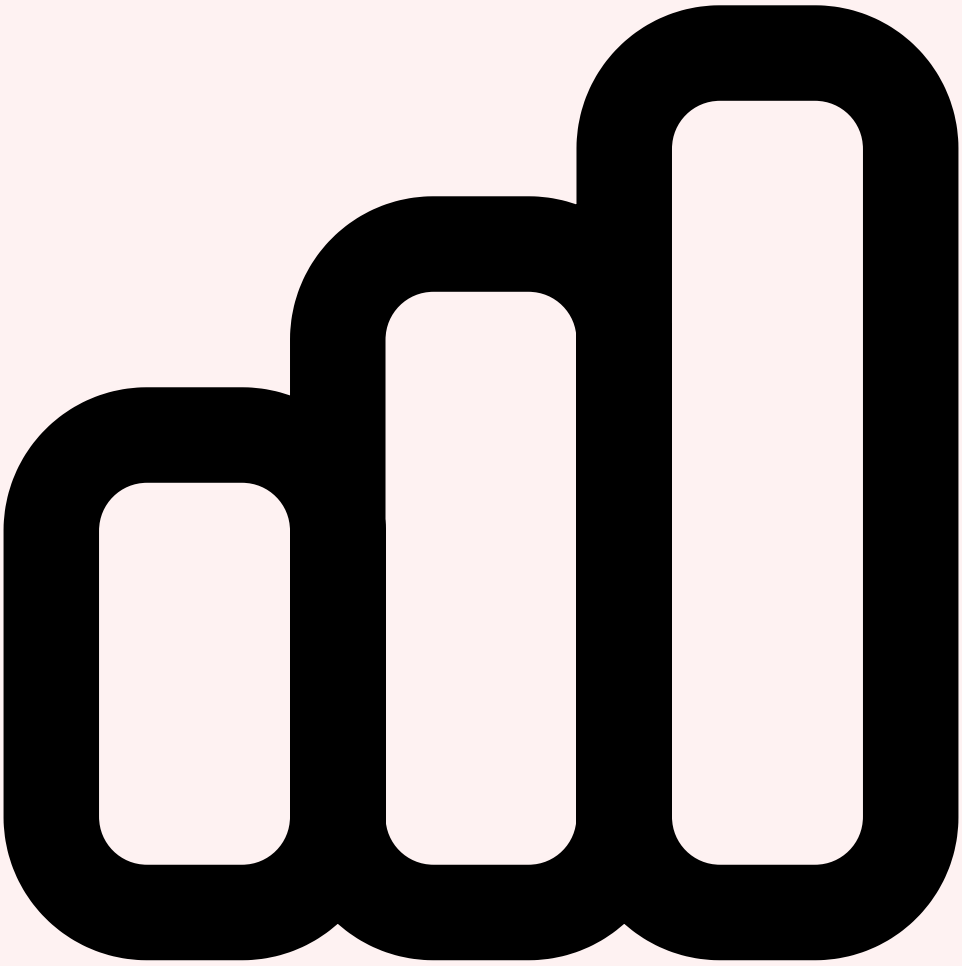
1 Les Défis du Déploiement de LLM en Production

Déployer un **Large Language Model** en production est un défi radicalement différent de l'entraînement ou de l'expérimentation en notebook. Si les benchmarks académiques mesurent la qualité des réponses, la réalité de la production impose des contraintes d'un tout autre ordre : **latence inférieure à 200 millisecondes pour le premier token**, débit soutenu de centaines de requêtes par seconde, disponibilité de 99,9 % et maîtrise d'une facture cloud qui peut atteindre des dizaines de milliers d'euros par mois pour un seul modèle. En 2026, la démocratisation des LLM open-weight — Llama 3.1 405B, Mistral Large 2, DeepSeek-V3, Qwen 2.5 72B — a rendu l'accès aux modèles de pointe plus facile que jamais, mais a simultanément complexifié les choix architecturaux. L'écart entre « ça fonctionne sur mon GPU de développement » et « ça tient 10 000 requêtes concurrentes en production » reste un gouffre que de nombreuses équipes sous-estiment.

La complexité intrinsèque des LLM en production

Le premier défi est la **taille des modèles** et les exigences mémoire qu'elle impose. Un modèle de 70 milliards de paramètres en précision FP16 nécessite environ 140 Go de VRAM, soit l'équivalent de deux GPU NVIDIA H100 avec 80 Go chacun. Le passage à des modèles de 400 milliards de paramètres demande un minimum de huit GPU haut de gamme, avec une interconnexion NVLink ou InfiniBand pour assurer un sharding efficace. Cette exigence mémoire ne concerne que les poids du modèle : le **KV-cache** (Key-Value cache), qui stocke les clés et valeurs d'attention pour chaque requête active, peut consommer des dizaines de gigaoctets supplémentaires sous forte charge. Pour un modèle 70B avec un context window de 128K tokens et 32 requêtes concurrentes, le KV-cache peut à lui seul exiger 50 à 80 Go de VRAM supplémentaires, dépassant souvent la mémoire allouée aux poids du modèle eux-mêmes.

Le second défi majeur est la **nature autogressive de la génération**. Contrairement à un modèle de classification ou de détection d'objets qui produit une réponse en un seul forward pass, un LLM génère du texte token par token. Chaque token nécessite un passage complet à travers le réseau, et la génération d'une réponse de 500 tokens implique 500 inférences séquentielles. Cette nature séquentielle crée un goulot d'étranglement fondamental : même avec des GPU surpuissants, la latence de génération est dominée par le nombre de tokens à produire et par la bande passante mémoire du GPU, pas par sa puissance de calcul brute. C'est pourquoi les architectures modernes de serving distinguent la phase de **prefill** (traitement du prompt d'entrée, intensive en calcul) de la phase de **decode** (génération token par token, intensive en bande passante mémoire). Cette distinction est central dans l'optimisation de l'inférence LLM en 2026.



Les contraintes opérationnelles et économiques

Au-delà des aspects techniques, les **contraintes opérationnelles** transforment un simple déploiement en un véritable projet d'infrastructure. La gestion des pannes GPU est un problème quotidien à l'échelle : sur un cluster de 64 GPU NVIDIA H100, les statistiques

montrent qu'une défaillance matérielle survient en moyenne toutes les deux à trois semaines. Le système de serving doit donc intégrer des mécanismes de failover automatique, de redistribution des requêtes et de rechargement des shards de modèle sans interruption de service. Les mises à jour de modèle — passage d'une version fine-tunée à une autre, déploiement d'un nouveau modèle — doivent s'effectuer en **blue-green deployment** ou en **canary release** pour éviter toute interruption. L'aspect économique est également critique : le coût horaire d'un nœud 8xH100 sur les principaux cloud providers dépasse 30 euros de l'heure, soit plus de 21 000 euros par mois en usage continu. Chaque pourcentage d'optimisation du taux d'utilisation GPU se traduit directement en économies substantielles. Les entreprises qui réussissent le déploiement de LLM en production en 2026 sont celles qui ont investi dans une stack d'observabilité dédiée, des pipelines de déploiement automatisés et une expertise profonde en infrastructure GPU — un triptyque que nous allons détailler dans les sections suivantes de cet article.

Point clé : Le déploiement de LLM en production est un problème d'infrastructure autant que de machine learning. Les trois défis majeurs sont la mémoire GPU (poids + KV-cache), la latence de génération autogressive et les coûts opérationnels qui peuvent atteindre 250 000 euros par an pour un seul modèle 70B en haute disponibilité.

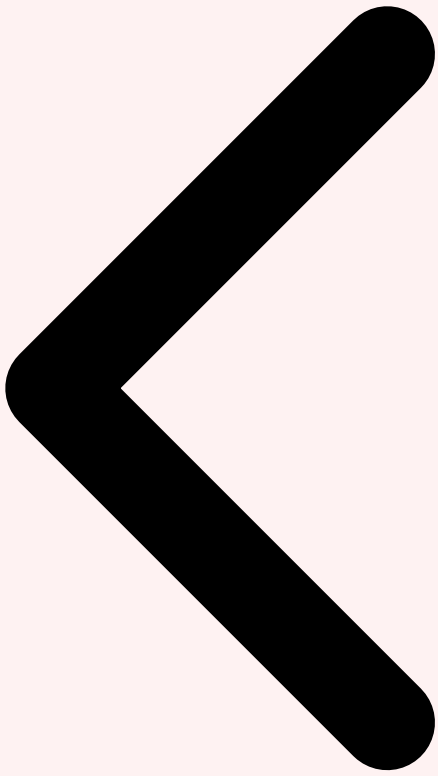
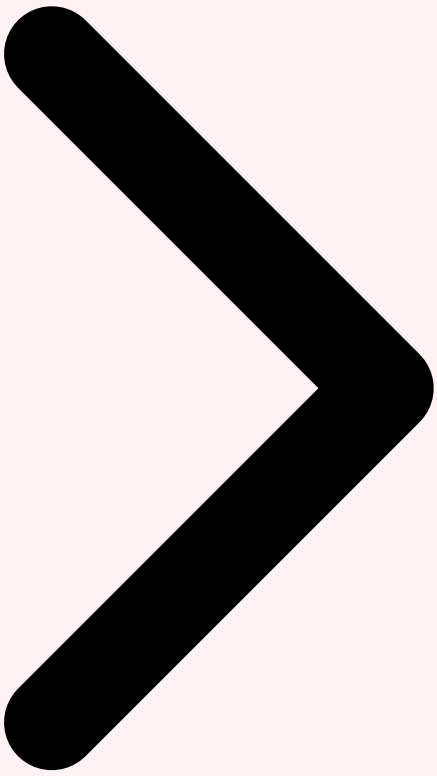


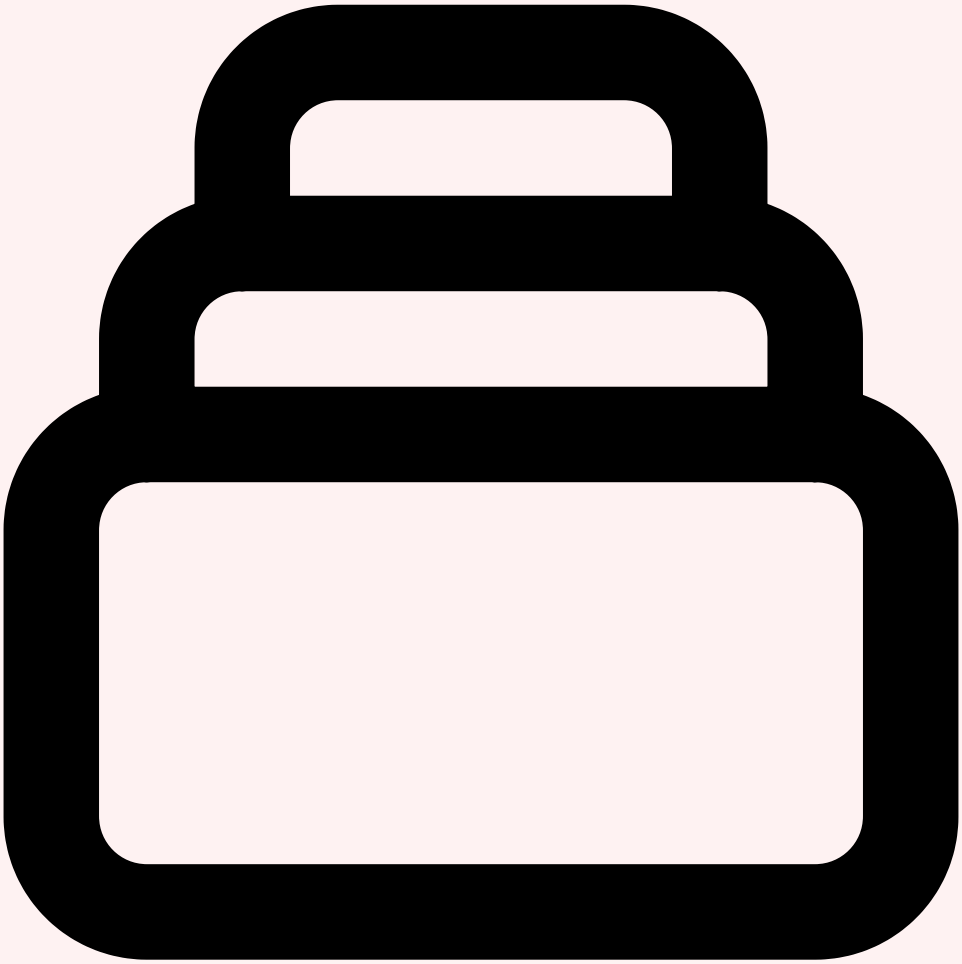
Table des Matières Défis Production LLM Architecture Serving



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.

2 Architecture de Serving LLM

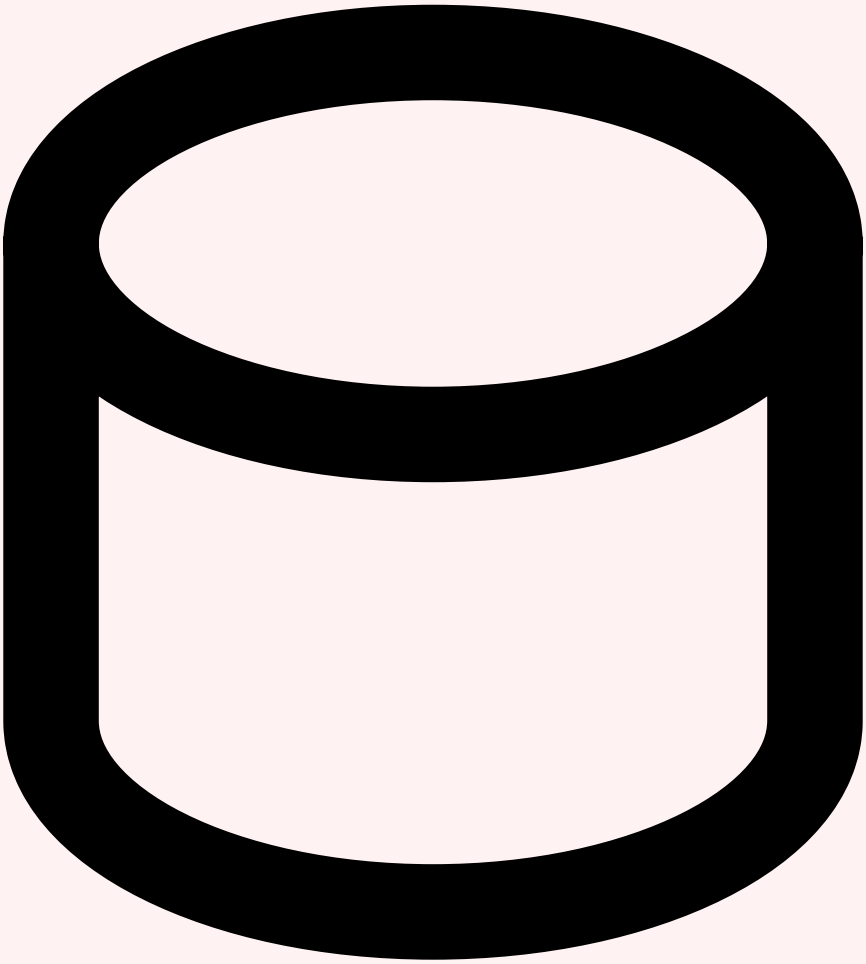
L'architecture de serving d'un LLM en production se décompose en plusieurs couches complémentaires, chacune répondant à des exigences spécifiques de performance, de fiabilité et de sécurité. En 2026, les architectures de référence ont convergé vers un modèle en quatre niveaux : **couche d'entrée** (API Gateway + Load Balancer), **couche de routage et scheduling** (Request Router + Continuous Batching Scheduler), **couche d'exécution GPU** (GPU Workers avec model sharding) et **couche d'infrastructure** (stockage de modèles, monitoring, autoscaling). Cette architecture en couches permet de découpler les préoccupations et d'optimiser indépendamment chaque composant, tout en maintenant une cohérence globale.



Couche d'entrée : API Gateway et Load Balancer

La couche d'entrée est le point de contact entre les clients et l'infrastructure de serving. Un **API Gateway** (Envoy, Kong, ou un gateway custom) gère l'authentification, le rate limiting par token bucket, la validation des requêtes et le routage vers les backends appropriés. Le rate limiting pour les LLM est spécifique : il ne suffit pas de limiter le nombre de requêtes par seconde, il faut également limiter le **nombre de tokens consommés** par utilisateur ou par clé API, car une requête avec un prompt de 100 000 tokens a un coût radicalement différent d'une requête de 100 tokens. Les implémentations modernes maintiennent des compteurs de tokens par fenêtre glissante et appliquent des quotas différenciés par niveau de service. Le load balancer distribue les requêtes entre les GPU workers en tenant compte de la charge réelle de chaque worker — non pas simplement le nombre de requêtes en cours, mais la **taille de la queue de tokens en attente** et le taux d'utilisation VRAM, informations remontées via des health checks spécialisés.

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

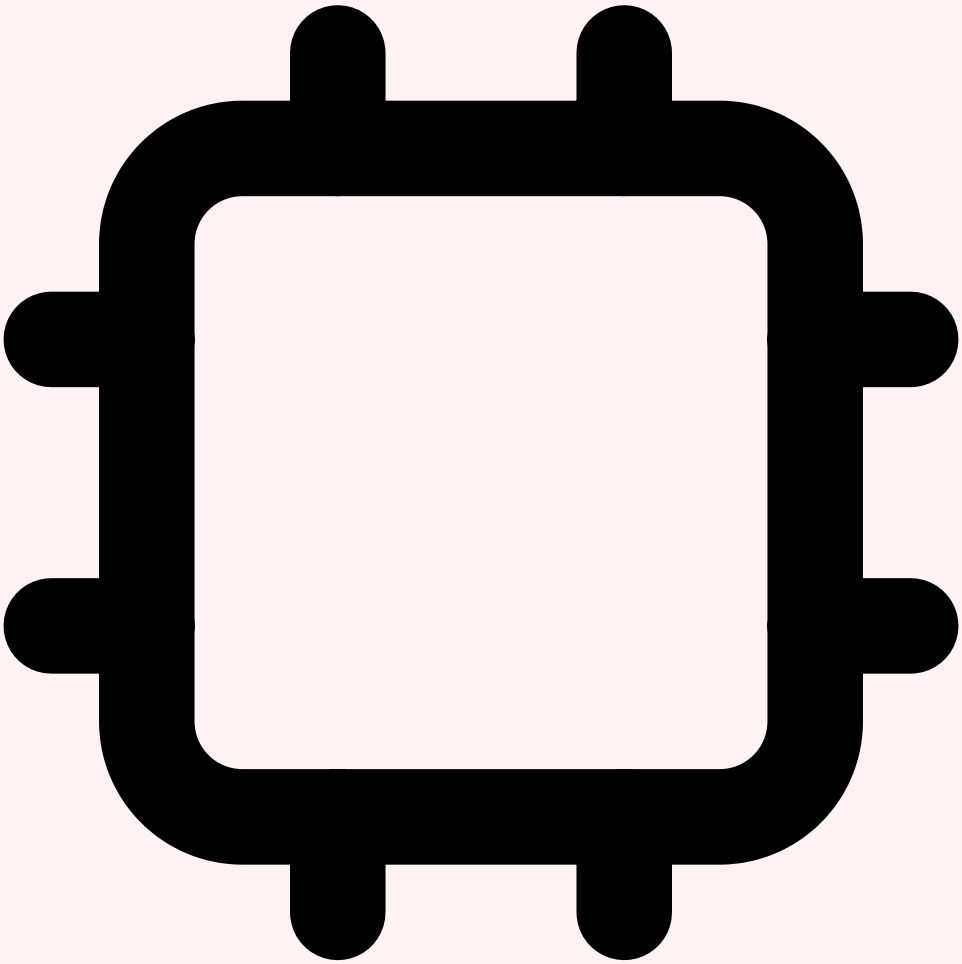


Couche de routage : Request Router et Continuous Batching

Le Request Router est le cerveau de l'architecture de serving. Son rôle est de **regrouper les requêtes en lots optimaux** (batches) et de les diriger vers les GPU workers les plus appropriés. Le **continuous batching**, introduit par Orca (Microsoft Research) et popularisé par vLLM, a transformé l'efficacité du serving LLM. Contrairement au batching statique traditionnel, qui attend qu'un lot complet de requêtes soit constitué avant de lancer l'inférence, le continuous batching insère dynamiquement de nouvelles requêtes dans un batch en cours dès qu'un slot se libère (lorsqu'une requête termine sa génération). Cette approche élimine les « bulles » d'inactivité GPU et peut multiplier le débit par un facteur 3 à 10 par rapport au batching statique. Le router intègre également le **prefix caching** : lorsque plusieurs requêtes partagent un même préfixe (par exemple, un system prompt identique), le KV-cache de ce préfixe est calculé une seule fois et réutilisé, réduisant considérablement la latence de prefill. En 2026, les routers avancés implémentent aussi le **speculative decoding routing**, qui dirige les requêtes simples vers des modèles draft légers et réserve les GPU haut de gamme pour les requêtes complexes nécessitant le modèle complet. Pour approfondir, consultez Milvus, Qdrant, Weaviate :.

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.

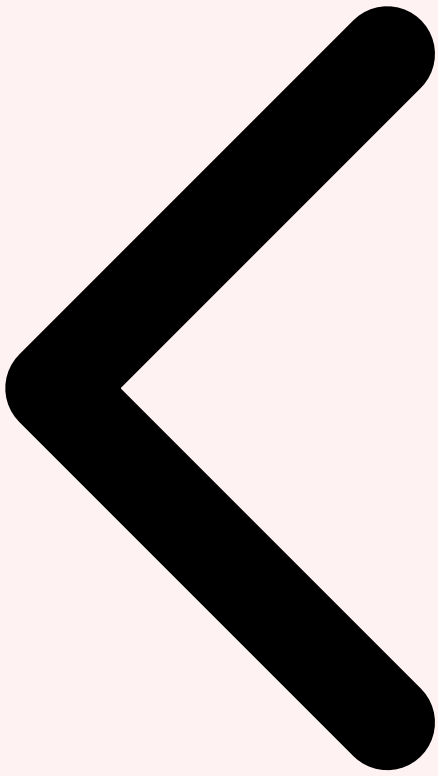


Couche d'exécution : GPU Workers et Model Sharding

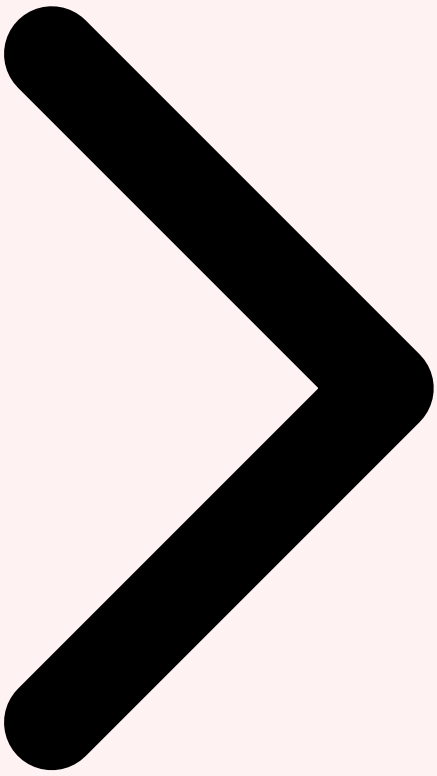
Les GPU workers constituent le cœur computationnel de l'architecture. Chaque worker charge une copie complète ou partielle du modèle et exécute l'inférence. Pour les modèles dépassant la capacité d'un seul GPU, le **model sharding** distribue les poids sur plusieurs GPU selon deux stratégies complémentaires. Le **Tensor Parallelism** (TP) découpe chaque couche du transformer en fragments distribués sur plusieurs GPU, chaque GPU calculant une portion des opérations d'attention et de feed-forward. Cette approche minimise la latence mais exige une interconnexion ultra-rapide (NVLink à 900 Go/s) car les GPU doivent échanger des activations à chaque couche. Le **Pipeline Parallelism** (PP) répartit les couches séquentiellement entre les GPU : le GPU 0 traite les couches 0-19, le GPU 1 les

couches 20-39, et ainsi de suite. Cette approche est moins exigeante en bande passante mais introduit des « micro-bulles » de latence. En pratique, les déploiements de 2026 combinent TP intra-nœud (GPU dans le même serveur, reliés par NVLink) et PP inter-nœuds (serveurs reliés par InfiniBand), optimisant ainsi le compromis entre latence et scalabilité. La gestion du **KV-cache** avec **PagedAttention** — inspirée de la gestion de mémoire virtuelle des systèmes d'exploitation — alloue la mémoire par pages de taille fixe plutôt qu'en blocs contigus, éliminant la fragmentation mémoire et augmentant le nombre de requêtes concurrentes de 2 à 4 fois par rapport aux approches naïves.

Architecture recommandée : Pour un modèle 70B en production : 2 GPU H100 par worker (TP=2), 3-4 workers derrière un load balancer avec continuous batching, prefix caching activé, et PagedAttention pour la gestion du KV-cache. Cette configuration supporte 200+ requêtes par seconde avec une latence P99 inférieure à 2 secondes.

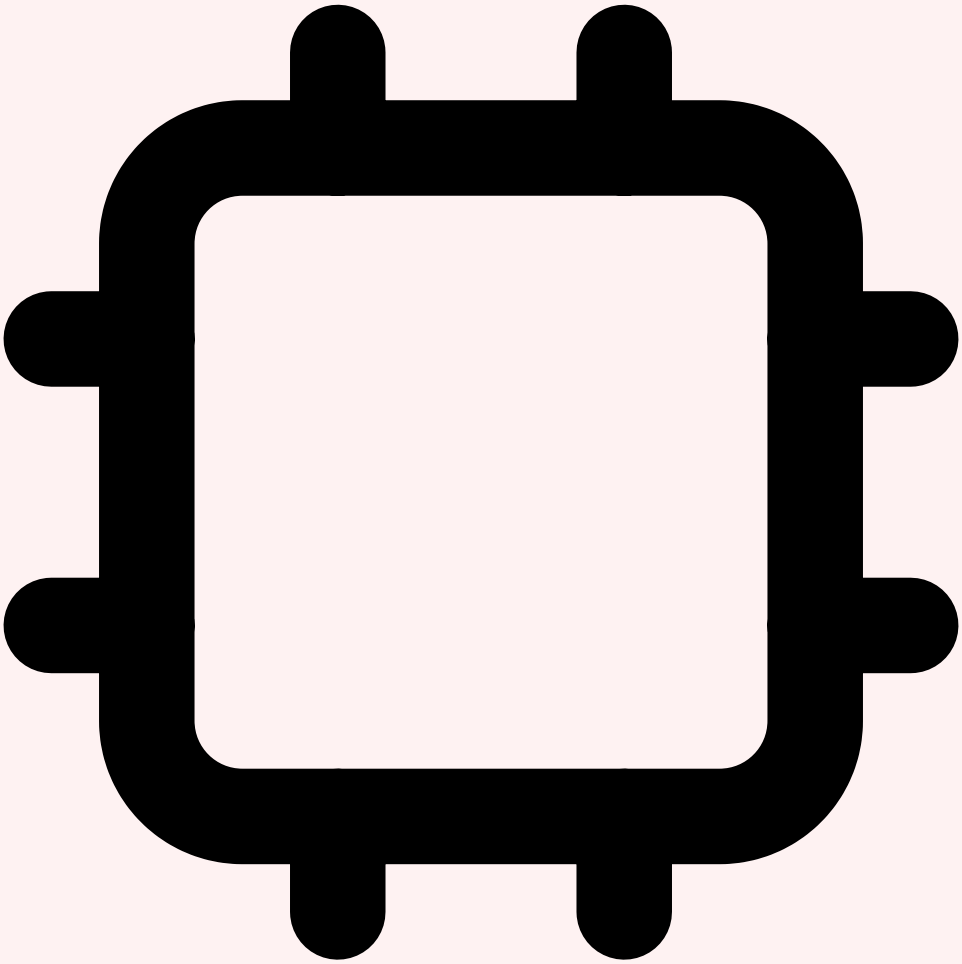


Défis Production LLM Architecture Serving Choix GPU



3 Choix du GPU : NVIDIA, AMD et Alternatives

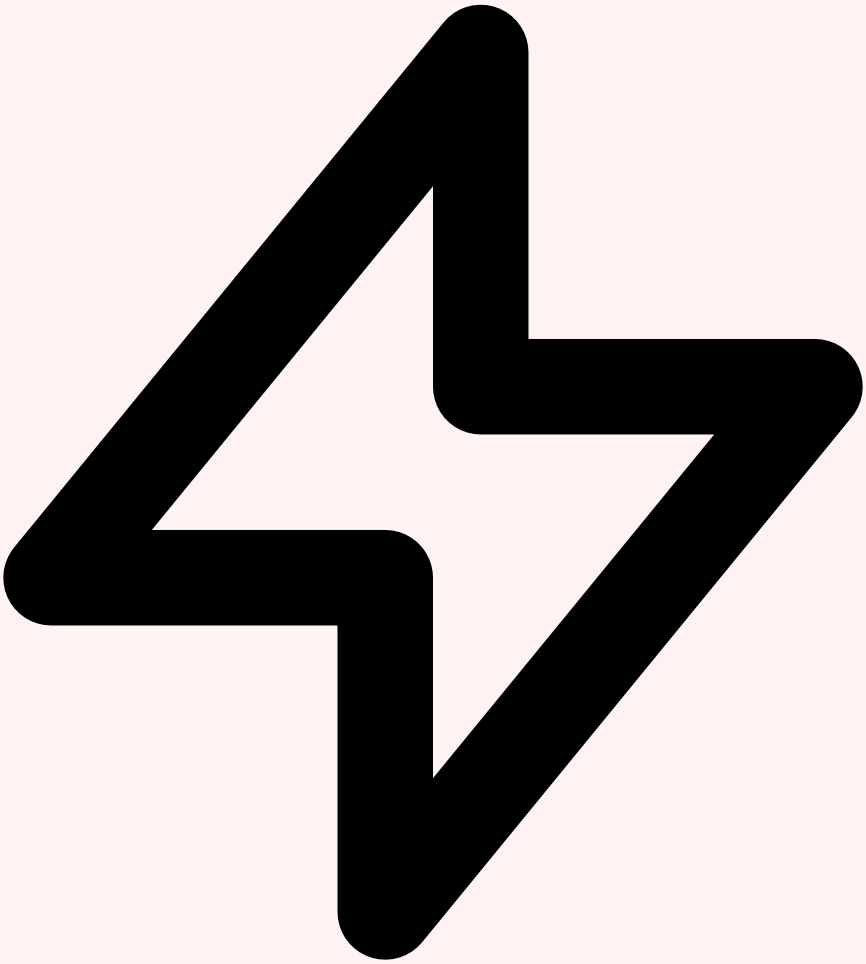
Le choix du GPU est la décision la plus structurante du déploiement d'un LLM en production. En 2026, le marché est dominé par **NVIDIA** avec plus de 80 % des déploiements d'inférence LLM, mais **AMD** et de nouveaux acteurs comme **Intel, Cerebras et Groq** commencent à proposer des alternatives crédibles pour des cas d'usage spécifiques. Le choix ne se résume pas à comparer les TFLOPS bruts : la **bande passante mémoire** (memory bandwidth), la **capacité VRAM**, l'**efficacité énergétique** et surtout la **maturité de l'écosystème logiciel** sont des critères au moins aussi importants pour l'inférence LLM.



NVIDIA : H100, H200 et Blackwell B200

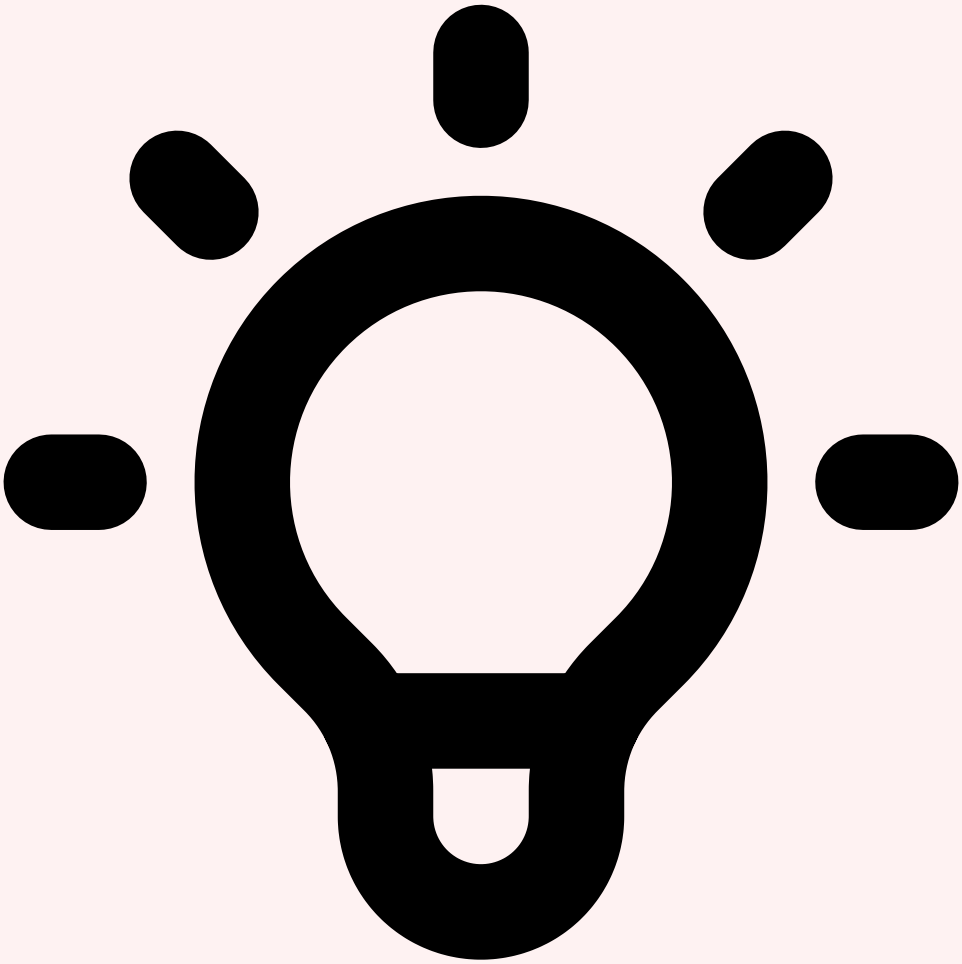
La gamme NVIDIA reste la référence incontournable pour l'inférence LLM. Le **H100 SXM**, avec ses 80 Go de HBM3 et une bande passante mémoire de 3,35 To/s, est le cheval de bataille de la majorité des déploiements en production depuis 2023. Sa performance en inférence LLM est de l'ordre de 1 500 à 2 500 tokens par seconde pour un modèle 70B en FP16 avec tensor parallelism sur deux GPU. Le **H200**, lancé en 2024, porte la mémoire à 141 Go de HBM3e avec une bande passante de 4,8 To/s — une augmentation de 43 % de la bande passante qui se traduit directement en gains de performance pour la phase de decode, bandwidth-bound. Pour les modèles de plus de 100 milliards de paramètres, le H200 élimine le besoin de sharding qui était nécessaire sur H100, simplifiant considérablement l'architecture. La nouvelle génération **Blackwell B200**, disponible en volume depuis fin 2025, franchit un cap majeur avec 192 Go de HBM3e et une bande passante de 8 To/s, doublant la capacité d'inférence par rapport au H200. Le B200 introduit également le support natif du FP4, permettant de servir des modèles quantifiés à 4 bits avec une précision quasi identique au FP8, réduisant de moitié la mémoire nécessaire et doublant le débit d'inférence.

GPU	VRAM	Bande passante	FP16 TFLOPS	TDP	Prix Cloud/h	Tokens/s (70B)
H100 SXM	80 Go HBM3	3,35 To/s	990	700W	~3,5 EUR	~2 000
H200 SXM	141 Go HBM3e	4,8 To/s	990	700W	~4,5 EUR	~3 200
B200 SXM	192 Go HBM3e	8 To/s	2 250	1 000W	~6,5 EUR	~5 500
AMD MI300X	192 Go HBM3	5,3 To/s	1 307	750W	~3,0 EUR	~2 800
AMD MI325X	256 Go HBM3e	6 To/s	1 307	750W	~3,8 EUR	~3 500
Groq LPU	230 Mo SRAM	80 To/s (on-chip)	750 (INT8)	300W	Cloud only	~800 (API)



AMD MI300X/MI325X : le challenger crédible

AMD a réalisé une percée significative avec la série **Instinct MI300X**, offrant 192 Go de HBM3 et une bande passante mémoire de 5,3 To/s — supérieure au H100 sur ces deux métriques critiques pour l'inférence. Le successeur **MI325X**, disponible depuis le premier trimestre 2026, pousse la capacité à 256 Go de HBM3e avec 6 To/s de bande passante, ce qui permet de charger un modèle de 120 milliards de paramètres en FP16 sur un seul GPU, sans aucun sharding. L'écosystème logiciel AMD, longtemps son point faible, a considérablement mûri grâce au support natif dans **vLLM, TGI et SGLang** via ROCm 6.x. Les benchmarks indépendants montrent que le MI300X atteint 85 à 95 % des performances du H100 pour l'inférence LLM, à un coût cloud inférieur de 15 à 20 %. Cependant, des frictions subsistent : certaines optimisations comme les custom CUDA kernels de FlashAttention ne sont pas encore totalement portées sur ROCm, et le débogage est moins mature. Pour les organisations prêtes à investir dans l'intégration, AMD représente en 2026 le meilleur rapport performance-prix pour l'inférence de modèles de grande taille.

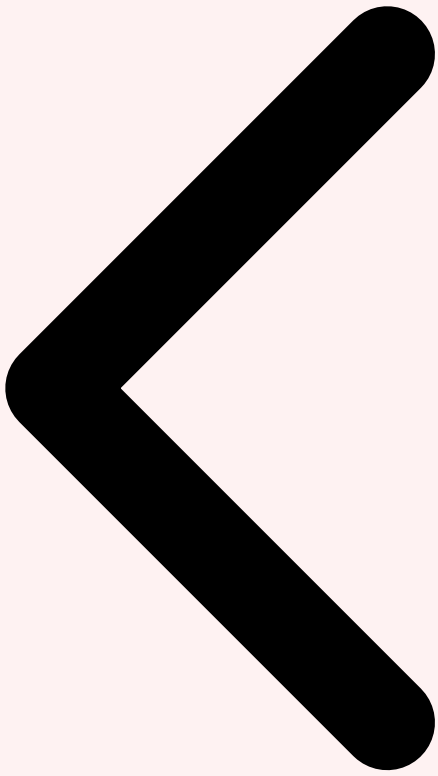


Alternatives émergentes : Groq, Cerebras et accélérateurs spécialisés

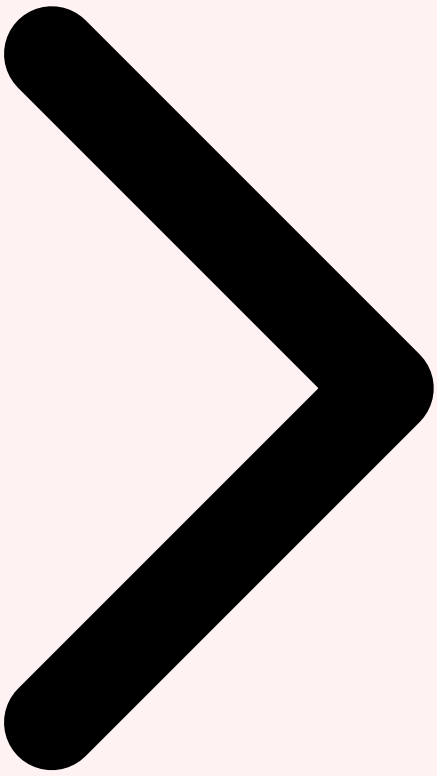
Les **accélérateurs spécialisés** représentent une approche fondamentalement différente de l'inférence LLM. **Groq** et son Language Processing Unit (LPU) utilisent une architecture à flux de données (dataflow) avec de la SRAM on-chip ultra-rapide au lieu de la HBM traditionnelle. Cette architecture élimine le goulot d'étranglement de la bande passante mémoire en maintenant tous les poids du modèle dans la SRAM distribuée, atteignant des latences de génération spectaculaires — jusqu'à 800 tokens par seconde pour des modèles de taille moyenne. La contrepartie est que la capacité SRAM limitée restreint la taille maximale des modèles supportés et que l'architecture nécessite un nombre important de puces pour les grands modèles. **Cerebras** avec son CS-3, basé sur un wafer-scale chip unique de 4 billions de transistors, offre une approche encore plus radicale : un seul chip peut contenir un modèle de 70 milliards de paramètres entièrement dans sa mémoire on-chip, éliminant totalement les communications inter-puces. **Intel Gaudi 3** propose une alternative plus conventionnelle mais avec un excellent rapport performance-prix, particulièrement adaptée aux déploiements on-premise dans des environnements réglementés. Le choix entre ces options dépend fondamentalement du profil de charge : pour un serving à haute concurrence avec des modèles de 70B+, les GPU NVIDIA ou AMD

restent le choix le plus polyvalent ; pour des applications à ultra-faible latence sur des modèles plus petits (7B-13B), les accélérateurs spécialisés peuvent offrir un avantage décisif.

Recommandation GPU 2026 : Pour un nouveau déploiement LLM en production, le **NVIDIA H200** offre le meilleur équilibre entre performance, maturité logicielle et disponibilité cloud. Le **B200** est le choix optimal si le budget le permet et la disponibilité le permet. Le **MI300X/MI325X** d'AMD est recommandé pour les organisations souhaitant diversifier leur dépendance fournisseur ou maximiser le rapport performance-prix.

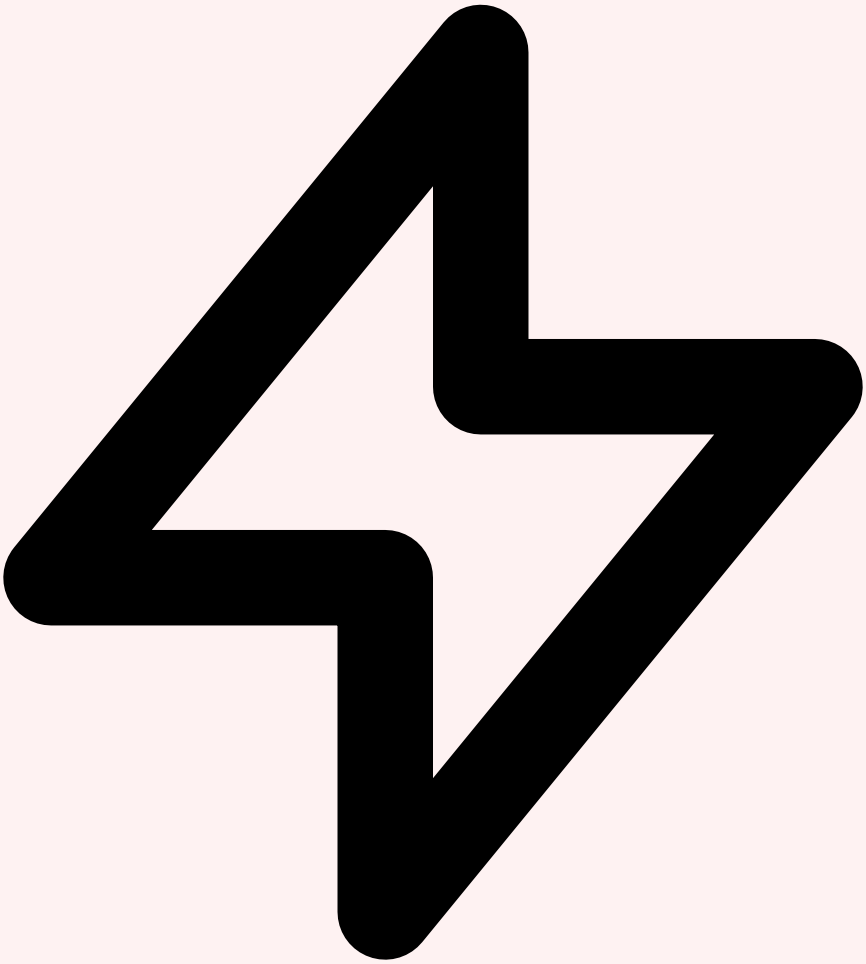


Architecture Serving Choix GPU Frameworks Serving



4 Frameworks de Serving : vLLM, TGI, SGLang

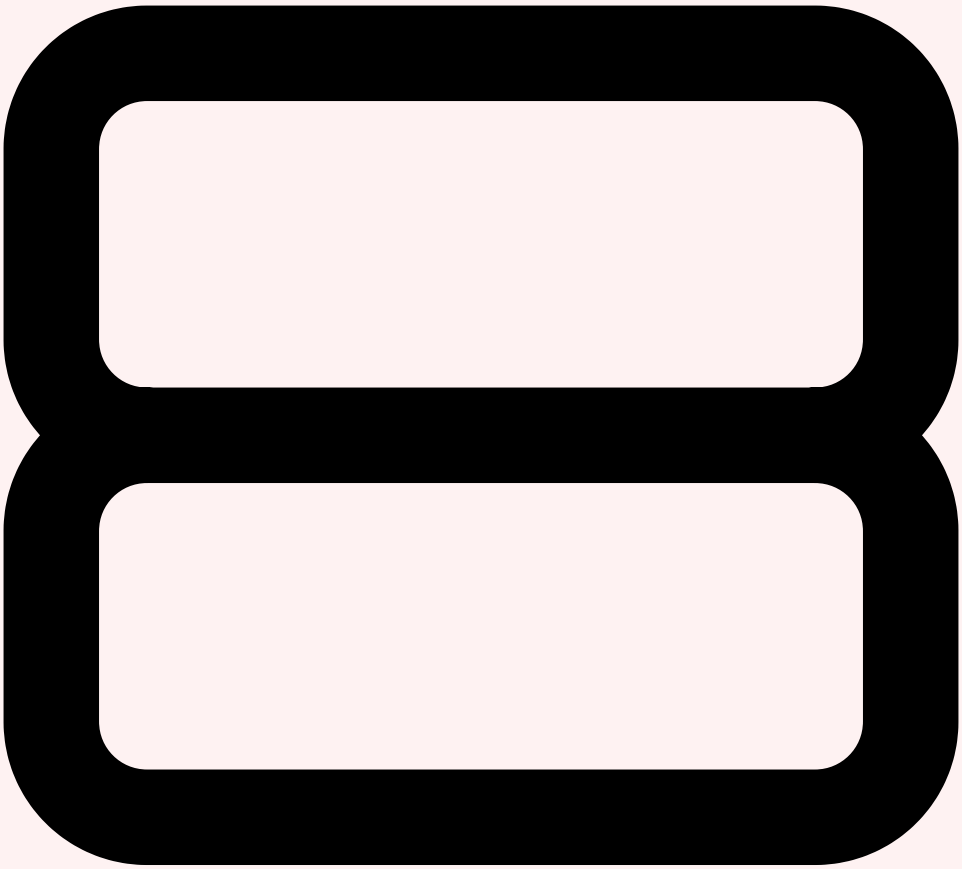
Le choix du framework de serving détermine directement les performances, la maintenabilité et l'évolutivité du déploiement. En 2026, trois frameworks open source se partagent la majorité des déploiements de production : **vLLM** (UC Berkeley), **Text Generation Inference (TGI)** (Hugging Face) et **SGLang** (UC Berkeley / Stanford). Chacun adopte une philosophie et des optimisations différentes, et le choix optimal dépend du profil de charge, du modèle utilisé et des contraintes opérationnelles de l'équipe.



vLLM : le standard de facto

vLLM s'est imposé comme le framework de serving LLM le plus adopté en production, grâce à son innovation fondatrice : **PagedAttention**. Ce mécanisme, inspiré de la pagination mémoire des systèmes d'exploitation, gère le KV-cache de manière non contiguë en pages de taille fixe, éliminant la fragmentation mémoire qui limitait le nombre de requêtes concurrentes dans les frameworks précédents. En 2026, vLLM v0.7+ intègre nativement le **continuous batching**, le **prefix caching** (réutilisation du KV-cache pour les préfixes partagés), le **speculative decoding** (utilisation d'un modèle draft léger pour accélérer la génération) et le support multi-GPU avec tensor parallelism et pipeline parallelism. L'API de vLLM est compatible OpenAI, ce qui facilite la migration depuis les API commerciales. Le déploiement type consiste à lancer un serveur vLLM derrière un reverse proxy Nginx avec plusieurs replicas orchestrés par Kubernetes. Les performances sont remarquables : sur un nœud 2xH100, vLLM peut servir un modèle Llama 3.1 70B à plus de 200 requêtes par seconde avec une latence P50 inférieure à 500 ms pour le premier token. Les limitations incluent une configuration parfois complexe pour les scénarios multi-nœuds

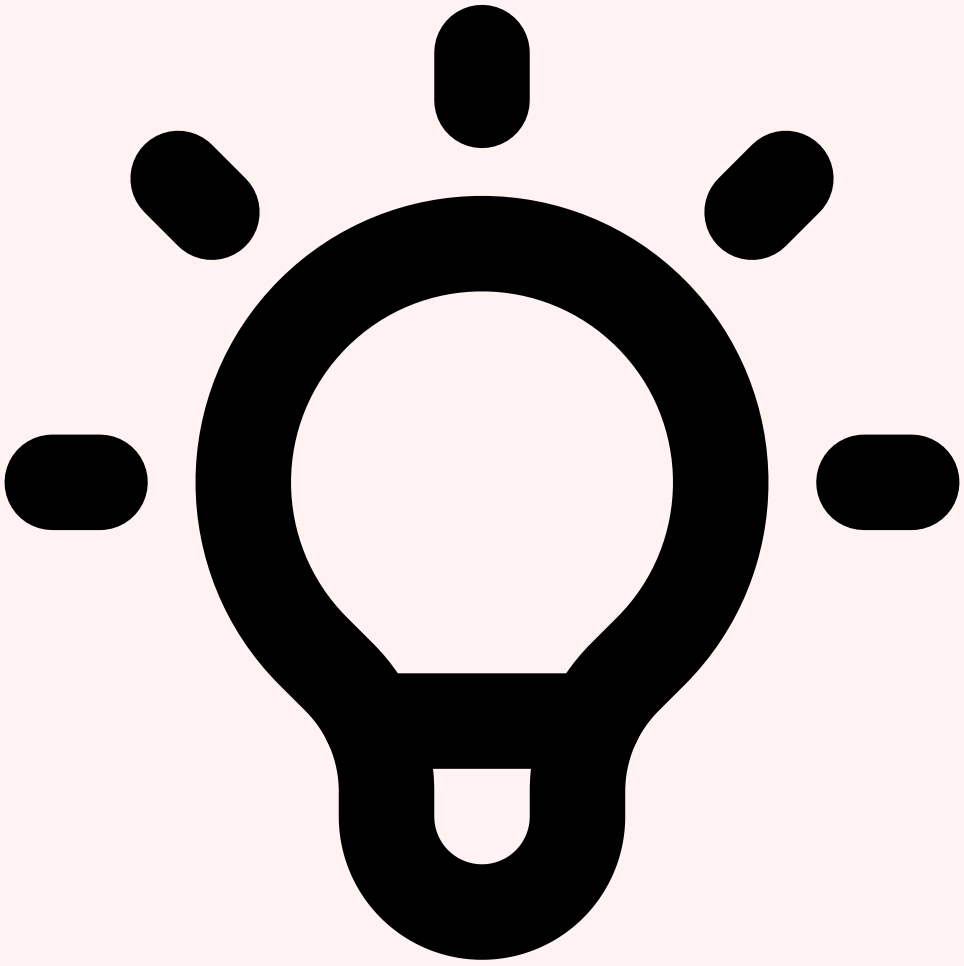
et un overhead mémoire pour la gestion des pages KV-cache, qui consomme environ 5 % de la VRAM disponible. Pour approfondir, consultez Quantum Machine Learning : Risques et Opportunités pour la.



TGI : l'approche intégrée de Hugging Face

Text Generation Inference (TGI) de Hugging Face se distingue par son **intégration native avec l'écosystème Hugging Face**. Écrit en Rust avec des kernels CUDA optimisés, TGI offre des performances comparables à vLLM tout en simplifiant considérablement le déploiement pour les utilisateurs de l'écosystème HF. Le chargement d'un modèle depuis le Hub se fait en une seule ligne de commande, avec détection automatique de l'architecture et application des optimisations appropriées (FlashAttention 2, quantification GPTQ/AWQ/EETQ, tensor parallelism). TGI v2.x (2026) intègre le **FlashDecoding** pour accélérer la phase de decode sur les séquences longues, le **chunked prefill** qui permet de découper les longs prompts en chunks traités progressivement sans bloquer la génération des autres requêtes, et le support des **modèles multimodaux** (vision-language models comme LLaVA et Qwen-VL). L'architecture Rust de TGI lui confère un avantage en termes de fiabilité et de

gestion mémoire : les fuites mémoire sont quasi inexistantes, ce qui est critique pour les déploiements long-running en production. TGI est également le backend par défaut des **Inference Endpoints** de Hugging Face, offrant une solution clé en main pour les équipes ne souhaitant pas gérer leur propre infrastructure. La contrepartie est une flexibilité moindre pour les configurations avancées et un rythme d'adoption des dernières innovations légèrement plus lent que vLLM.



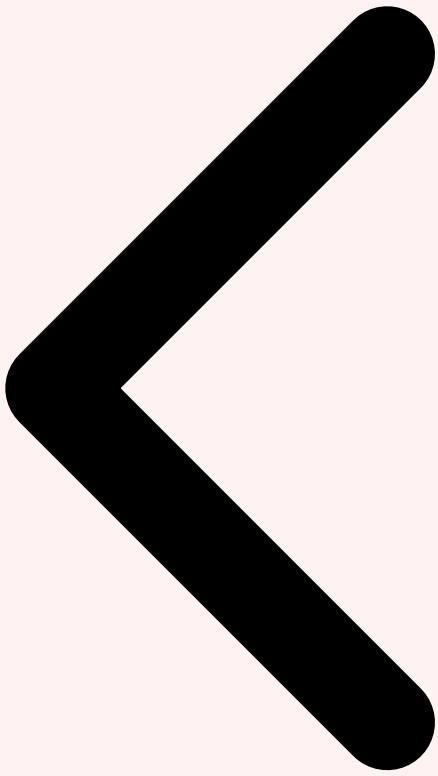
SGLang : la nouvelle frontière du serving structuré

SGLang (Structured Generation Language) représente l'approche la plus innovante du serving LLM en 2026. Développé par les équipes de UC Berkeley et Stanford, SGLang se distingue par sa capacité à **optimiser les programmes LLM structurés** — des flux de travail impliquant plusieurs appels au modèle avec des contraintes de format (JSON, regex, grammaires). Là où vLLM et TGI traitent chaque requête indépendamment, SGLang comprend les relations entre les appels successifs et optimise le KV-cache en conséquence, avec un système de **RadixAttention** qui maintient un arbre de préfixes permettant de réutiliser massivement le cache entre les requêtes apparentées. Pour les pipelines d'agents IA qui enchaînent extraction, raisonnement et génération structurée, SGLang peut être 3 à 5 fois plus rapide que vLLM grâce à cette optimisation cross-requêtes. SGLang v0.4+ intègre

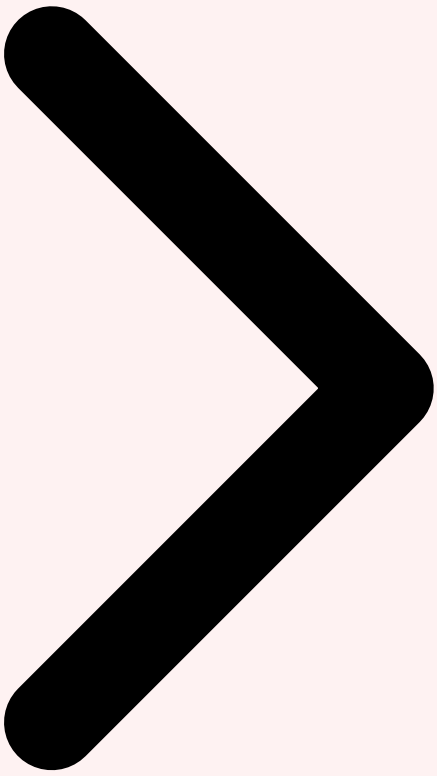
également un **moteur de génération contrainte ultra-performant** basé sur des automates finis, capable de forcer le respect d'un schéma JSON ou d'une grammaire BNF avec un overhead de seulement 2 à 5 % sur la vitesse de génération, contre 20 à 30 % pour les approches concurrentes. L'adoption en production est encore moindre que vLLM ou TGI, mais SGLang gagne rapidement du terrain, particulièrement dans les déploiements d'agents IA et les applications nécessitant une sortie structurée fiable. La roadmap 2026 inclut le support natif du speculative decoding et l'intégration avec les frameworks d'orchestration comme LangGraph et CrewAI.

Critère	vLLM	TGI	SGLang
Langage	Python + CUDA	Rust + CUDA	Python + Triton
Innovation clé	PagedAttention	FlashDecoding + Chunked Prefill	RadixAttention
Continuous Batching	Natif	Natif	Natif
Prefix Caching	Oui (APC)	Basique	Avancé (Radix)
Sortie structurée	Via outlines	Basique	Natif, ultra-rapide
Speculative Decoding	Oui	Oui	En développement
Support AMD ROCm	Oui	Oui	Expérimental
Maturité production	Excellente	Excellente	Bonne (en progrès)
Cas d'usage idéal	Serving généraliste haute performance	Écosystème HF, déploiement rapide	Agents, génération structurée

Choix pragmatique : Commencez par **vLLM** pour un serving généraliste — c'est le choix le plus sûr avec la plus grande communauté et le support le plus large. Migrez vers **SGLang** si votre application repose fortement sur les agents IA ou la génération structurée. Utilisez **TGI** si votre équipe est déjà investie dans l'écosystème Hugging Face et valorise la simplicité de déploiement.

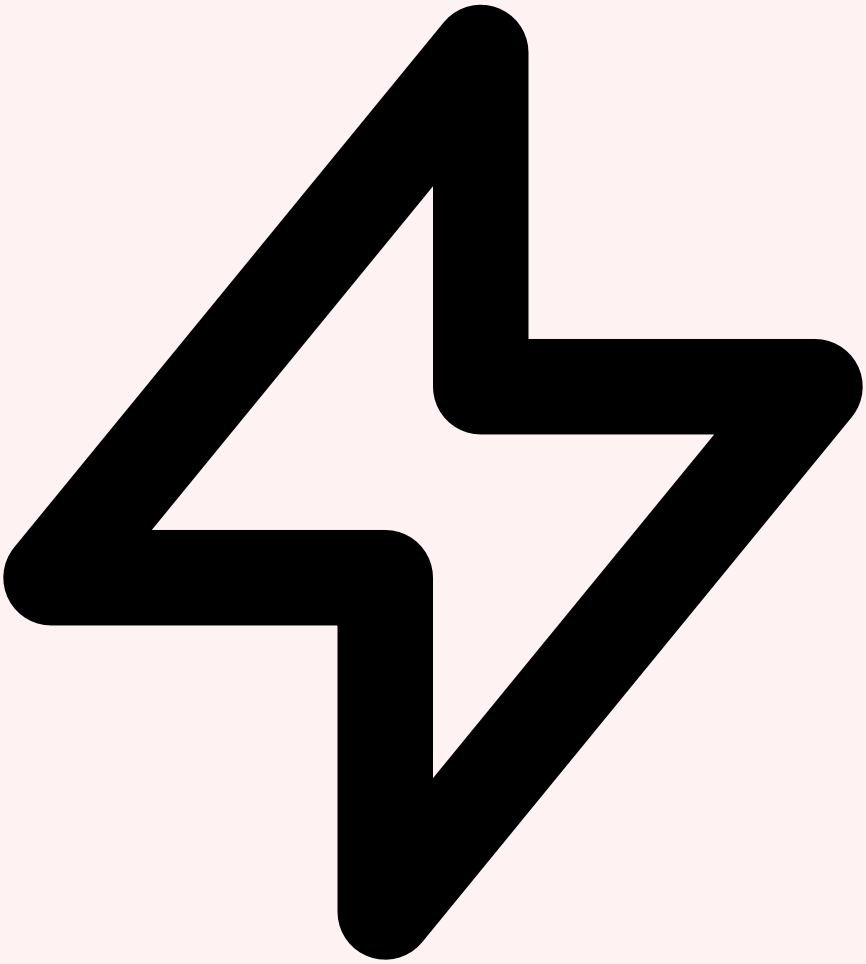


Choix GPU Frameworks Serving Optimisation Inférence



5 Optimisation de l'Inférence

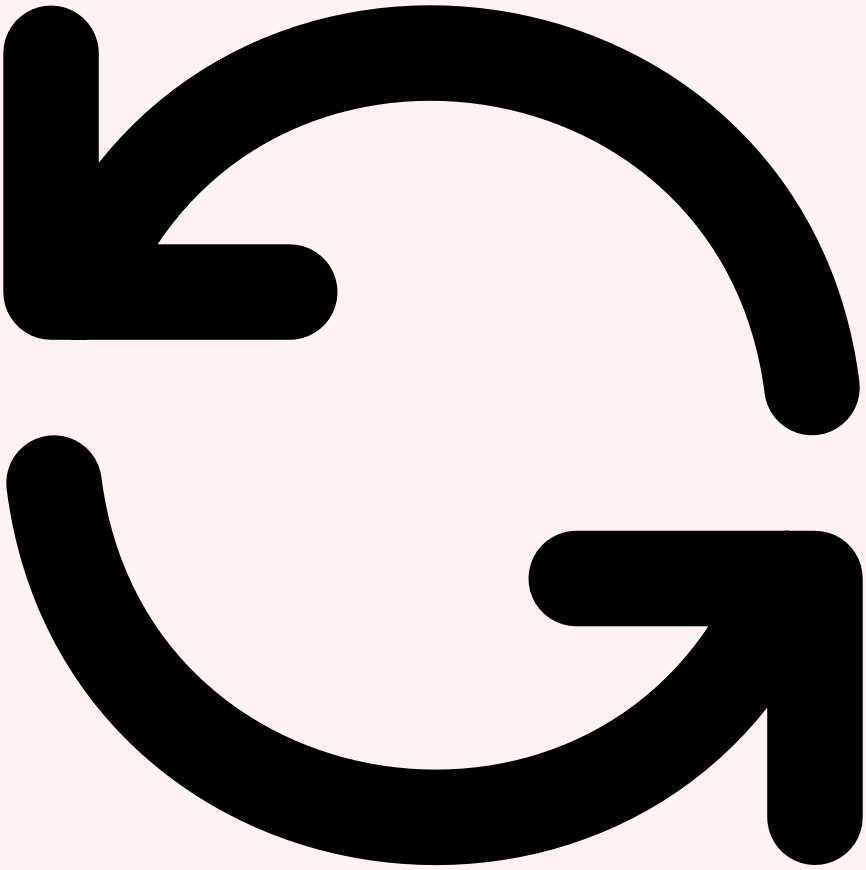
L'optimisation de l'inférence LLM est un domaine en évolution rapide où chaque amélioration se traduit directement en réduction de latence, augmentation du débit et économies de coûts. En 2026, les techniques d'optimisation se répartissent en trois catégories : la **quantification des poids et des activations**, les **optimisations algorithmiques** (FlashAttention, speculative decoding, chunked prefill) et les **optimisations système** (compilation de graphes, kernels custom, gestion mémoire avancée). La combinaison de ces techniques peut multiplier par 5 à 10 le débit d'inférence par rapport à une implémentation naïve en FP16, tout en maintenant une qualité de sortie quasi identique.



Quantification : FP8, INT4 et FP4

La **quantification** est la technique d'optimisation ayant le plus grand impact en production. Le principe est de réduire la précision numérique des poids du modèle — de FP16 (16 bits) à FP8 (8 bits), INT4 (4 bits) ou même FP4 (4 bits flottants) — tout en préservant la qualité des prédictions. En 2026, la quantification **FP8** est considérée comme « gratuite » : les GPU NVIDIA Hopper et Blackwell disposent d'unités de calcul FP8 natives, et la perte de qualité est inférieure à 0,1 % sur les benchmarks standard. La quantification FP8 réduit la mémoire de 50 % et double le débit d'inférence. Pour aller plus loin, les techniques **GPTQ, AWQ et SqueezeLLM** permettent une quantification à 4 bits (INT4) avec des pertes de qualité mesurables mais acceptables pour la plupart des cas d'usage (1 à 3 % de dégradation sur les benchmarks). L'approche **AWQ** (Activation-Aware Weight Quantization) est particulièrement populaire en 2026 car elle identifie les canaux « saillants » — les poids les plus critiques pour la qualité — et les préserve en précision supérieure, réduisant la dégradation à moins de 1 % même en INT4. La nouveauté de 2026 est le **FP4** supporté nativement par les GPU Blackwell B200, offrant les avantages de la

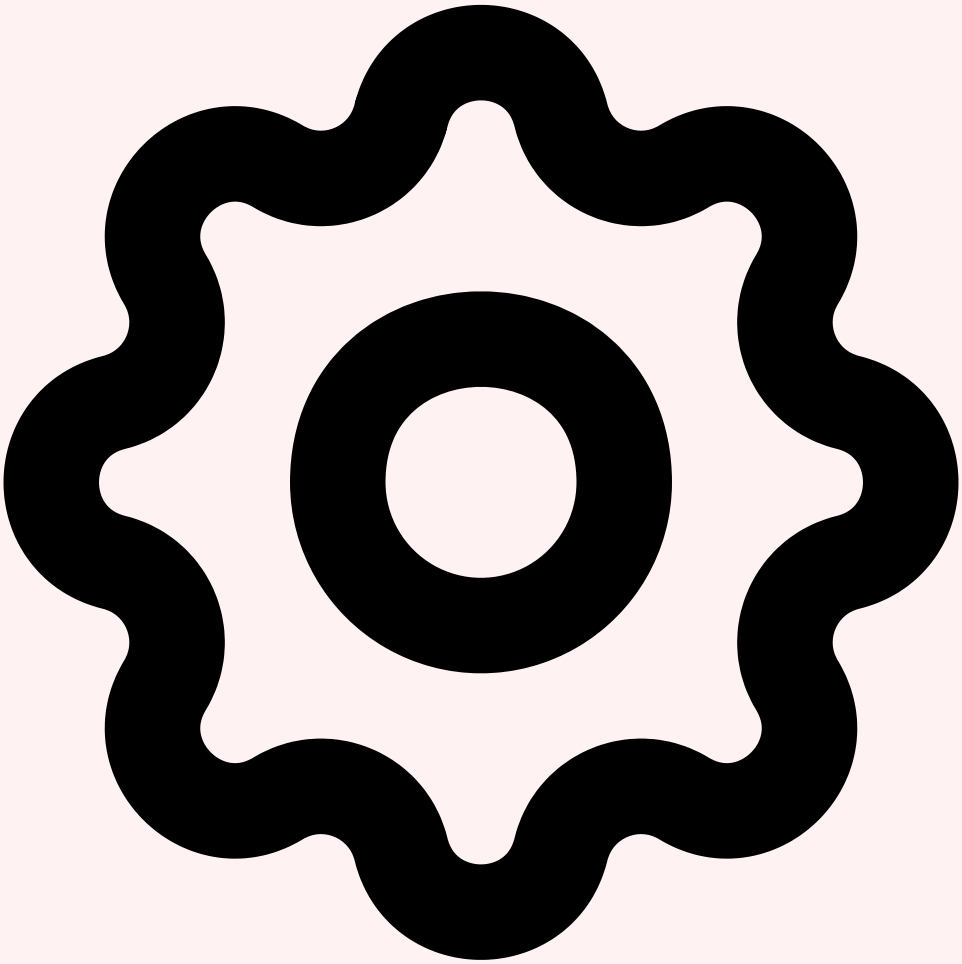
quantification 4 bits avec une meilleure préservation de la dynamique des valeurs que l'INT4. La chaîne d'optimisation typique en 2026 est : entraînement en BF16, puis quantification post-entraînement en FP8 ou AWQ-INT4 pour le serving.



Speculative Decoding et Chunked Prefill

Le **speculative decoding** est une technique qui utilise un petit modèle « draft » (typiquement 1 à 7 milliards de paramètres) pour générer rapidement une séquence candidate de tokens, puis le modèle principal vérifie cette séquence en un seul forward pass parallèle. Si les tokens générés par le draft model sont corrects (ce qui se produit dans 60 à 80 % des cas pour un bon draft model), plusieurs tokens sont validés en un seul pas d'inférence du modèle principal, accélérant la génération de 2 à 3 fois sans aucune perte de qualité. La difficulté réside dans le choix du draft model : il doit être suffisamment rapide pour ne pas annuler le gain (latence du draft < latence d'un token du modèle principal) et suffisamment aligné avec le modèle principal pour maximiser le taux d'acceptation. Les implémentations de 2026 utilisent des techniques comme le **Medusa** (ajout de têtes de prédiction parallèles au modèle principal) et le **Eagle** (draft model entraîné par distillation

du modèle principal), qui atteignent des taux d'acceptation de 75 à 85 %. Le **chunked prefill**, quant à lui, résout un problème différent : lorsqu'une requête avec un très long prompt (50 000+ tokens) arrive, le prefill peut bloquer l'ensemble du batch pendant plusieurs secondes. Le chunked prefill découpe ce long prefill en chunks de 512 à 2 048 tokens, traités progressivement entre les steps de decode des autres requêtes, éliminant les pics de latence pour les requêtes concurrentes.

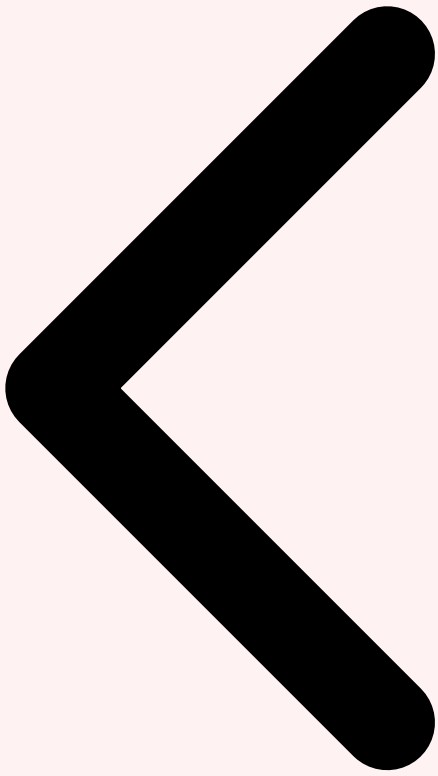


Compilation de graphes et kernels custom

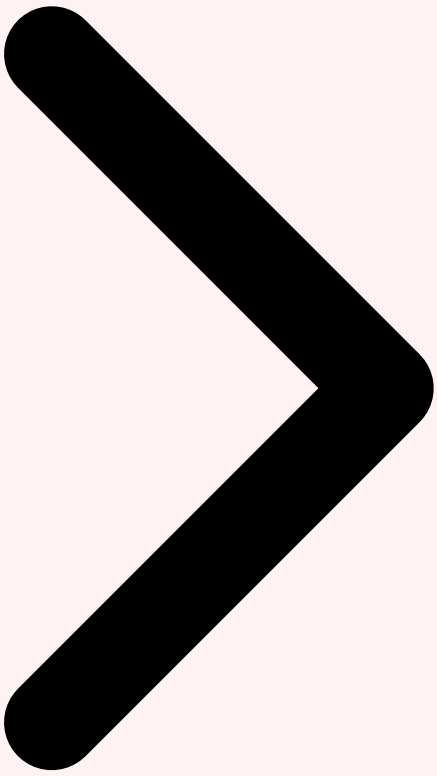
La dernière couche d'optimisation concerne la **compilation et les kernels GPU custom**. **torch.compile** (PyTorch 2.x) fusionne automatiquement les opérations GPU, éliminant les allers-retours CPU-GPU et les allocations mémoire intermédiaires. Sur l'inférence LLM, torch.compile peut apporter un gain de 10 à 30 % de débit, particulièrement sur la phase de prefill. Les **kernels Triton** (OpenAI) permettent d'écrire des kernels GPU haute performance en Python, démocratisant l'optimisation GPU auparavant réservée aux experts CUDA. **FlashAttention 3**, disponible sur les GPU Hopper et Blackwell, exploite l'asynchronisme matériel et le pipelining des opérations de mémoire et de calcul pour atteindre une utilisation de 75 % des FLOPS théoriques du GPU sur les opérations d'attention, contre 30 à 40 % pour les implémentations standard. Pour les déploiements à

très haut débit, **TensorRT-LLM** de NVIDIA compile le modèle en un graphe d'exécution optimisé avec fusion de couches, quantification automatique et exécution parallèle des opérations indépendantes. TensorRT-LLM v0.15+ (2026) offre des gains de 20 à 50 % par rapport à vLLM en mode natif PyTorch, au prix d'un temps de compilation de 15 à 45 minutes et d'une flexibilité réduite pour les modèles custom.

Chaîne d'optimisation recommandée : Étape 1 : Quantification FP8 (gratuite, +100% débit). **Étape 2** : PagedAttention + continuous batching via vLLM (+300% concurrence). **Étape 3** : Prefix caching pour les system prompts partagés (+30% latence P50). **Étape 4** : Speculative decoding si la latence de génération est critique (+150% vitesse). **Étape 5** : TensorRT-LLM si le modèle est stable et le débit maximal est requis (+30% supplémentaires). Pour approfondir, consultez Automatiser le DevOps avec des Agents IA : Guide Complet.

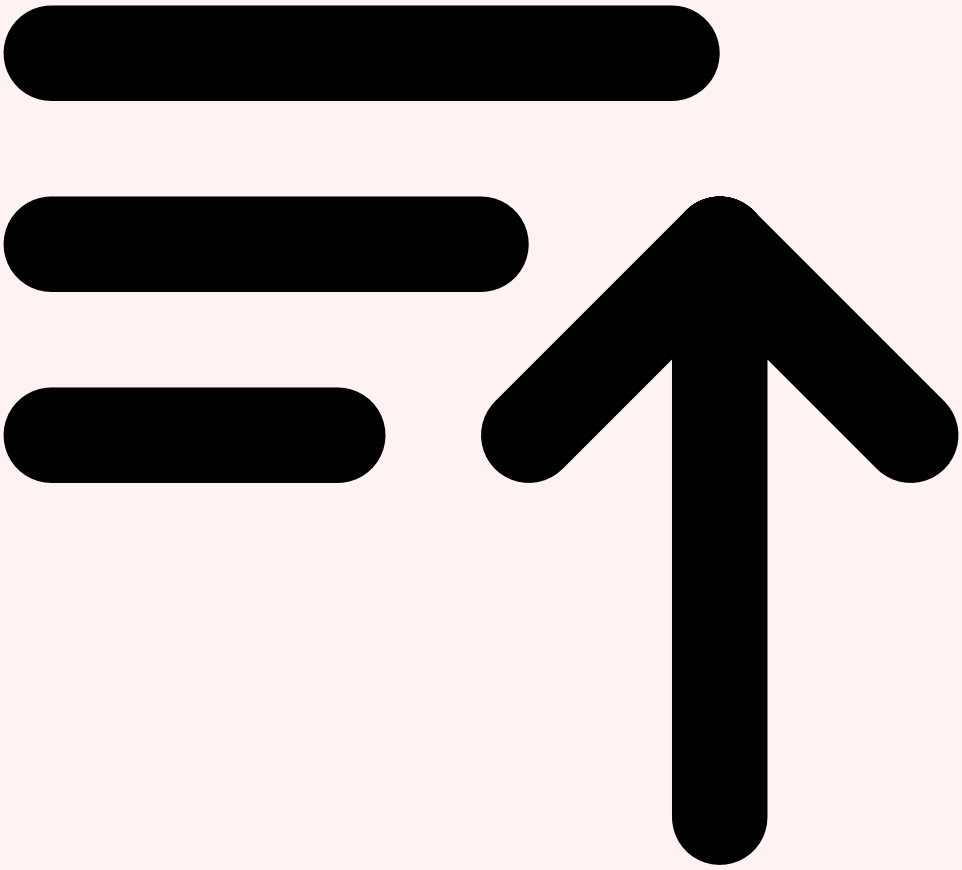


Frameworks Serving Optimisation Inférence Scaling Production



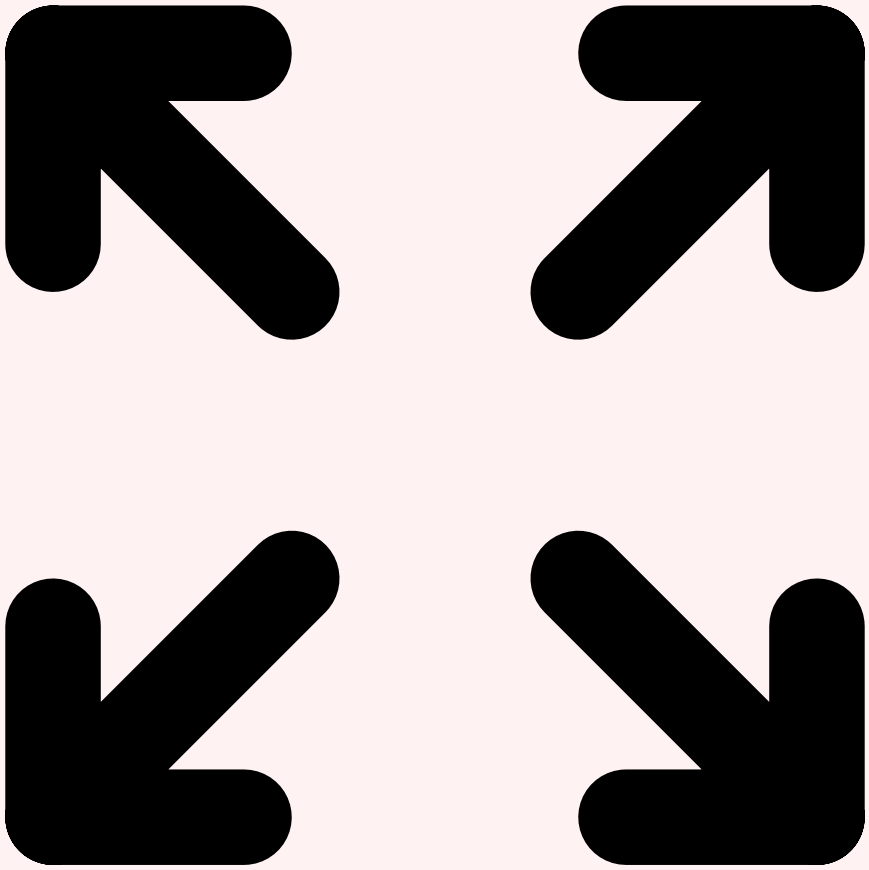
6 Scaling en Production : Horizontal et Vertical

Le scaling d'un service LLM en production est un exercice d'équilibriste entre performance, coût et complexité opérationnelle. Contrairement aux services web traditionnels où le scaling horizontal est quasi linéaire (ajouter un serveur double la capacité), le **scaling d'un LLM** est contraint par la taille du modèle, les exigences de cohérence du KV-cache et le coût prohibitif des GPU. En 2026, les architectures de production les plus performantes combinent judicieusement scaling vertical (optimiser chaque worker au maximum) et scaling horizontal (répliquer les workers), avec des stratégies avancées comme le **disaggregated serving** et le **routage intelligent multi-modèle**.



Scaling vertical : maximiser chaque nœud GPU

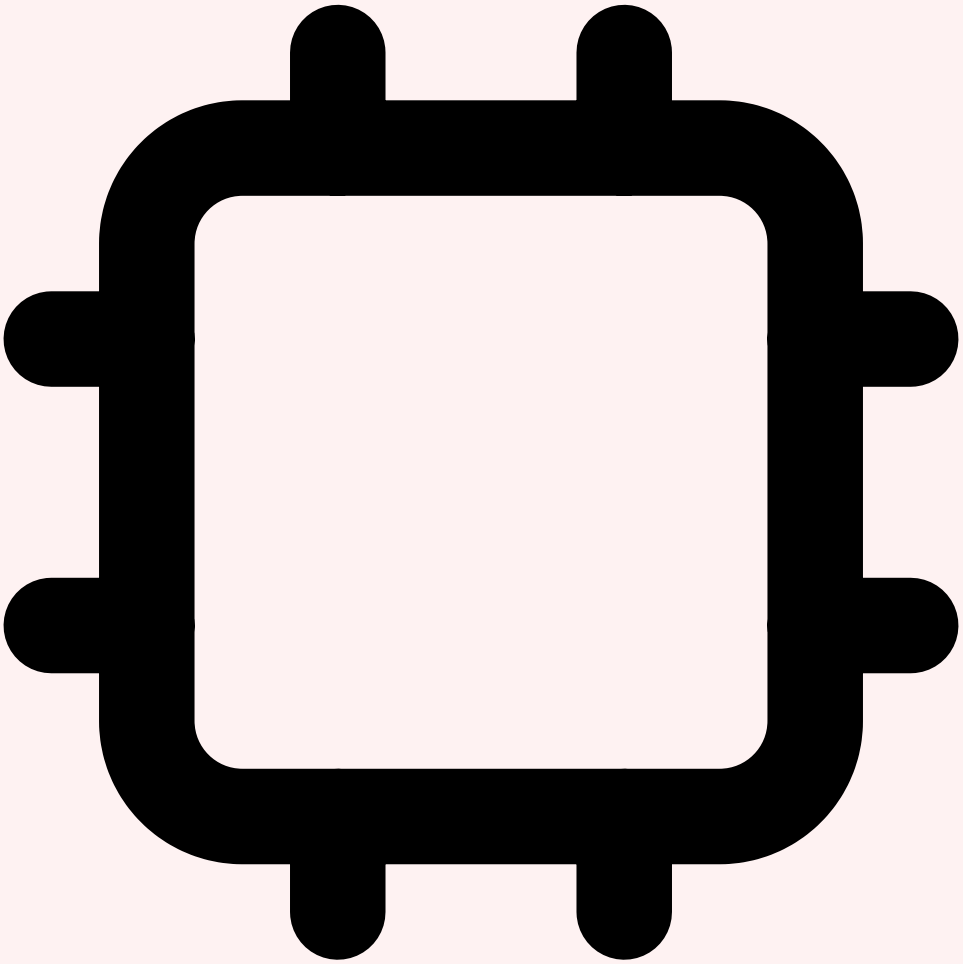
Le scaling vertical vise à extraire le maximum de performance de chaque nœud GPU avant d'ajouter des nœuds supplémentaires. La première étape est le choix du **degré de tensor parallelism** optimal. Pour un modèle 70B, la configuration TP=2 sur deux H100 80Go offre le meilleur rapport débit/coût : chaque GPU traite la moitié des opérations de chaque couche, la communication NVLink à 900 Go/s est suffisante pour ne pas devenir un goulot d'étranglement, et le modèle tient confortablement en mémoire avec de l'espace pour le KV-cache. Passer à TP=4 réduit la latence d'environ 30 % (grâce au parallélisme accru) mais double le coût GPU et introduit des overheads de synchronisation supplémentaires. La règle empirique est : **utiliser le degré de TP minimal qui permet au modèle de tenir en mémoire avec suffisamment d'espace pour le KV-cache cible**. Pour un modèle 70B en FP8 avec un contexte de 32K tokens et 32 requêtes concurrentes, TP=2 sur H100 80Go est optimal. Pour le même modèle avec un contexte de 128K tokens, TP=4 est nécessaire pour accommoder le KV-cache élargi. La quantification joue un rôle central dans le scaling vertical : un modèle 70B quantifié en AWQ-INT4 ne nécessite que 35 Go de VRAM pour les poids, permettant un serving sur un seul GPU H100 avec un KV-cache confortable — passant de TP=2 à TP=1 divise le coût par deux.



Scaling horizontal : réplication et routage intelligent

Une fois chaque worker optimisé verticalement, le scaling horizontal augmente la capacité globale en **répliquant les workers derrière un load balancer**. La stratégie de load balancing pour le LLM serving est spécifique : un simple round-robin est inefficace car les requêtes ont des coûts de traitement très variables (un prompt de 100 tokens vs 100 000 tokens). Les load balancers modernes pour LLM utilisent le **least-pending-tokens** : chaque worker remonte en temps réel le nombre de tokens en cours de traitement, et le load balancer dirige les nouvelles requêtes vers le worker le moins chargé. Le **routage multi-modèle** est une stratégie avancée qui maintient plusieurs tailles de modèle en production simultanément : un modèle 7B pour les tâches simples (classification, extraction d'entités), un modèle 70B pour les tâches intermédiaires (résumé, traduction, Q&A), et un modèle 405B pour les tâches complexes (raisonnement, code generation avancée). Un router intelligent analyse chaque requête et la dirige vers le modèle le plus adapté, optimisant le ratio coût/qualité. Les implémentations de 2026 utilisent un petit modèle classifieur (ou des heuristiques basées sur la longueur du prompt et les paramètres de la requête) pour

effectuer ce routage en moins d'une milliseconde. Cette approche permet de réduire le coût d'inférence de 50 à 70 % par rapport à l'utilisation systématique du modèle le plus puissant, avec une dégradation de qualité perceptible par l'utilisateur inférieure à 5 %.

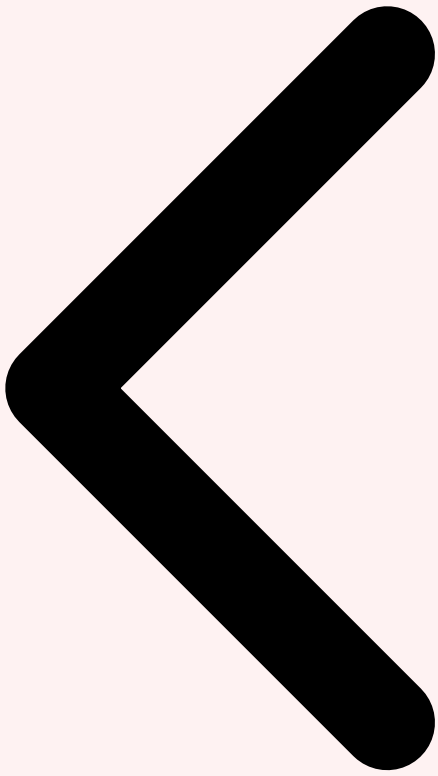


Disaggregated serving et autoscaling GPU

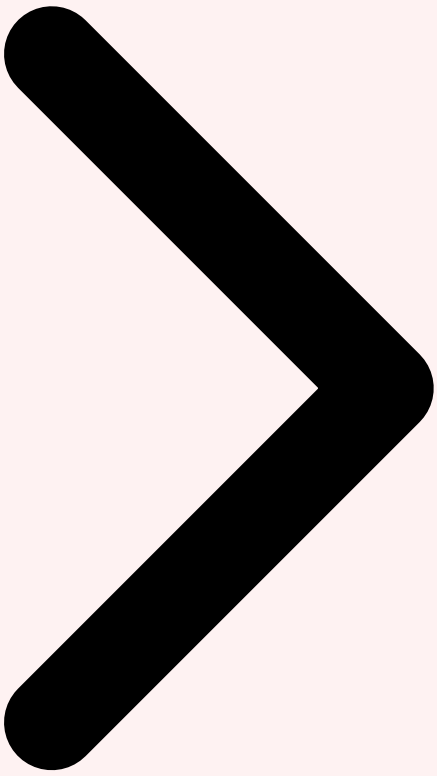
Le **disaggregated serving** est l'innovation architecturale la plus prometteuse de 2026 pour le scaling LLM. Le principe est de séparer physiquement les **prefill workers** (qui traitent les prompts d'entrée) des **decode workers** (qui génèrent les tokens de sortie). Cette séparation est justifiée par le fait que le prefill est **compute-bound** (il bénéficie de GPU à haute puissance de calcul) tandis que le decode est **memory-bandwidth-bound** (il bénéficie de GPU à haute bande passante mémoire). En pratique, cela signifie que les prefill workers peuvent utiliser des GPU optimisés pour le calcul (comme le H100 avec ses tensor cores) tandis que les decode workers peuvent utiliser des GPU avec plus de mémoire et de bande passante (comme le H200 ou le MI300X). Le KV-cache calculé pendant le prefill est transféré au decode worker via un réseau haute vitesse (RDMA sur InfiniBand ou NVLink inter-nœuds). Les benchmarks montrent que le disaggregated serving peut améliorer le débit global de 30 à 50 % par rapport au serving monolithique, en permettant un dimensionnement indépendant des capacités de prefill et de decode selon le profil de

charge. L'**autoscaling GPU** avec **KEDA** (Kubernetes Event-Driven Autoscaling) et les métriques GPU custom (utilisation SM, utilisation mémoire, longueur de queue) permet un scaling élastique en 30 à 60 secondes. Le défi principal est le **GPU prewarming** : le chargement d'un modèle 70B sur un nouveau nœud prend 2 à 5 minutes, ce qui nécessite des stratégies prédictives de pré-provisioning basées sur les patterns de trafic historiques.

Stratégie de scaling recommandée : Phase 1 (0-100 req/s) : Un seul worker optimisé (TP=2, FP8, PagedAttention). **Phase 2** (100-500 req/s) : 3-5 workers identiques avec load balancing least-pending-tokens. **Phase 3** (500+ req/s) : Disaggregated serving + routage multi-modèle + autoscaling KEDA avec GPU prewarming prédictif. Chaque phase doit être validée en charge avant de passer à la suivante.

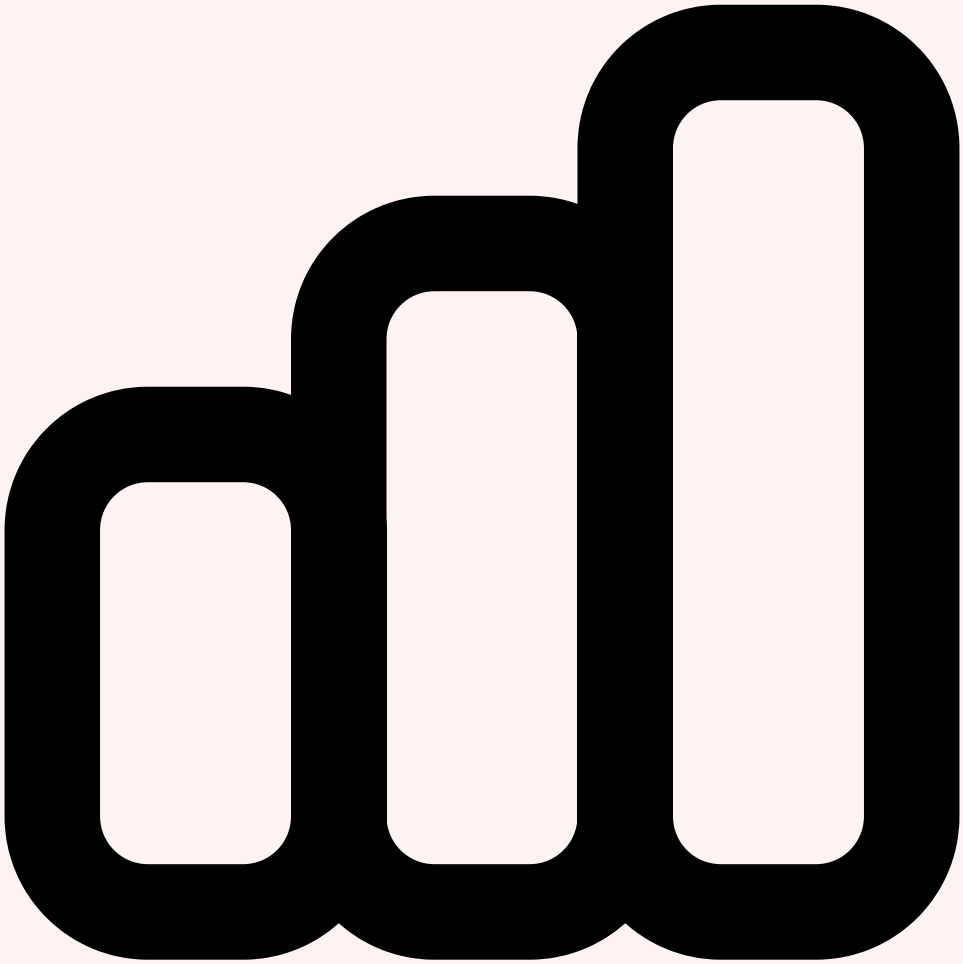


Optimisation Inférence Scaling Production Monitoring



7 Monitoring et Observabilité

Le monitoring d'un service LLM en production diffère fondamentalement du monitoring applicatif traditionnel. Les métriques classiques — latence HTTP, taux d'erreur, CPU — sont insuffisantes pour comprendre la santé d'un système de serving LLM. En 2026, les équipes de production les plus matures ont développé un **stack d'observabilité à trois niveaux** : les métriques de performance d'inférence (latence par token, débit, utilisation GPU), les métriques de qualité de service (longueur de queue, taux de timeout, taux de rejet) et les métriques de coût et d'efficacité (coût par token, utilisation GPU effective, ratio compute/bandwidth). Cette observabilité multi-dimensionnelle est indispensable pour diagnostiquer les problèmes de performance, optimiser les coûts et planifier la capacité.



Métriques essentielles du serving LLM

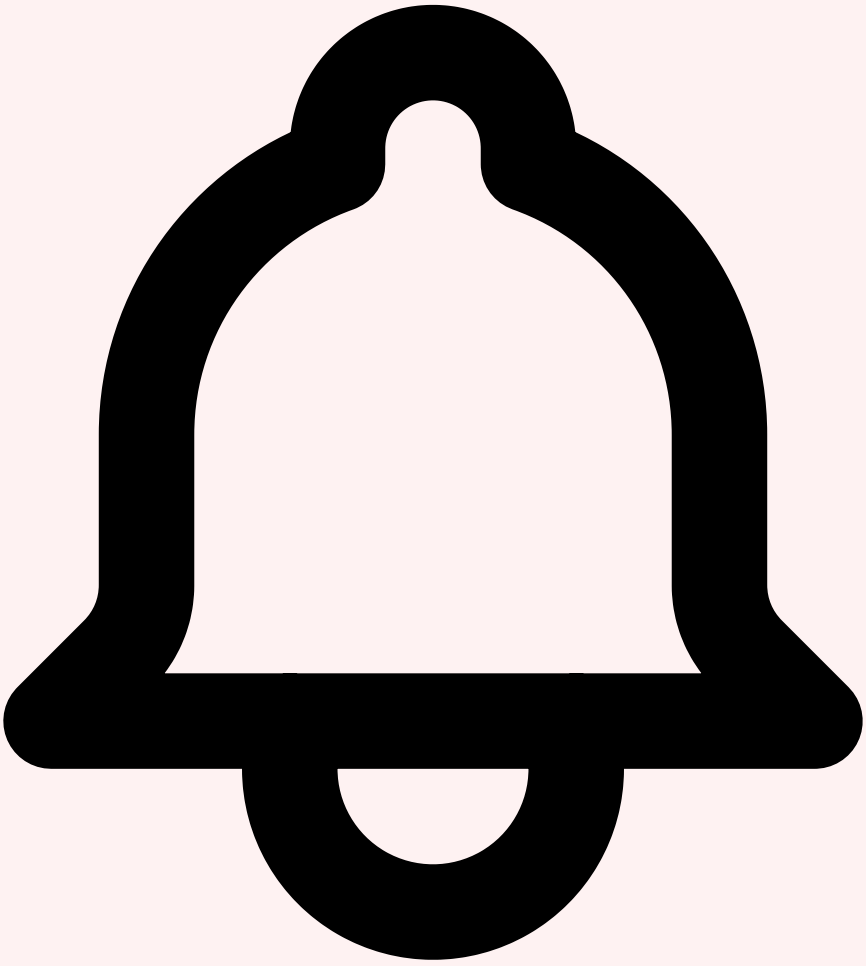
Les métriques de performance d'inférence LLM se décomposent en plusieurs catégories critiques. Le **Time To First Token (TTFT)** mesure la latence entre la réception de la requête et l'émission du premier token de la réponse — c'est la métrique perçue par l'utilisateur comme « le temps de réflexion ». Le TTFT dépend principalement de la phase de prefill et de la longueur du prompt d'entrée. Un SLO typique est un TTFT P99 inférieur à 2 secondes. Le **Time Per Output Token (TPOT)** mesure l'intervalle entre chaque token généré après le premier — c'est la « vitesse de frappe » du modèle perçue par l'utilisateur. Un TPOT de 30 à 50 ms correspond à une vitesse de lecture confortable, tandis qu'un TPOT supérieur à 100 ms crée une perception de lenteur. Le **débit en tokens par seconde** (tous utilisateurs confondus) mesure la capacité globale du système. Le **taux d'utilisation GPU** se décompose en deux sous-métriques : l'utilisation des Streaming Multiprocessors (SM utilization, mesurée par DCGM Exporter) et l'**Model FLOPs Utilization (MFU)** qui rapporte les FLOPS effectivement utilisés pour l'inférence aux FLOPS théoriques du GPU. Un MFU de 40 à 60 % est considéré bon pour l'inférence LLM (inférieur à l'entraînement car la phase de decode est bandwidth-bound). Enfin, l'**utilisation mémoire VRAM** distingue la mémoire

occupée par les poids du modèle (fixe) et le KV-cache (variable selon le nombre de requêtes actives et la longueur des séquences) — un dashboard en temps réel de cette décomposition est indispensable pour le capacity planning.

Stack d'observabilité recommandé

Le stack d'observabilité de référence pour le serving LLM en 2026 repose sur **Prometheus + Grafana** pour la collecte et la visualisation des métriques, enrichi de composants spécifiques à l'infrastructure GPU. **NVIDIA DCGM Exporter** (Data Center GPU Manager) expose des métriques GPU détaillées — température, utilisation SM, bande passante mémoire, erreurs ECC, consommation électrique — directement dans Prometheus. Les frameworks de serving (vLLM, TGI, SGLang) exposent nativement des métriques Prometheus incluant le nombre de requêtes en cours, la longueur de la queue, le nombre de tokens traités par seconde et l'utilisation du KV-cache. Le **dashboard Grafana type** pour le serving LLM comprend six panneaux critiques : (1) TTFT et TPOT par percentile (P50, P95, P99) avec alertes sur les dépassements de SLO, (2) débit global en tokens/seconde avec superposition de la capacité maximale théorique, (3) utilisation GPU par worker avec décomposition compute/memory, (4) KV-cache utilization avec la frontière entre zone saine et zone de saturation, (5) longueur de queue des requêtes en attente avec le taux de rejet,

et (6) coût par million de tokens calculé en temps réel à partir du prix cloud horaire et du débit observé. Pour le tracing distribué, **OpenTelemetry** avec des spans custom pour chaque phase de l'inférence (tokenization, prefill, decode, detokenization) permet de diagnostiquer précisément les goulots d'étranglement. Les traces incluent les métadonnées de chaque requête : nombre de tokens en entrée, nombre de tokens générés, temps de prefill, temps de decode, worker assigné et GPU utilisé.



Alerting et gestion des incidents GPU

L'alerting pour le serving LLM doit être calibré avec précision pour éviter à la fois les faux positifs (qui créent de la fatigue d'alerte) et les faux négatifs (qui laissent passer des dégradations de service). Les **alertes critiques** incluent : TTFT P99 dépassant le SLO pendant plus de 5 minutes (indiquant une saturation du prefill), taux de rejet de requêtes supérieur à 1 % (indiquant un KV-cache saturé ou un nombre insuffisant de workers), erreurs ECC GPU uncorrected (indiquant une défaillance matérielle imminente), et température GPU dépassant 85 degrés Celsius (risque de throttling thermique). Les **alertes de warning** incluent : utilisation VRAM dépassant 90 % (risque de saturation imminente), longueur de queue dépassant 50 requêtes (dégradation de latence progressive), et MFU inférieur à 30 % (indication d'une configuration sous-optimale ou d'un problème de

batching). La gestion des **incidents GPU** est un aspect opérationnel critique : une défaillance GPU dans un setup TP=2 rend l'intégralité du worker indisponible, pas seulement la moitié de sa capacité. Les runbooks d'incident doivent inclure des procédures automatisées de migration de charge vers les workers restants, de notification de l'équipe infrastructure et de lancement d'un worker de remplacement. Les tests de chaos engineering — injection de pannes GPU simulées, saturation mémoire, latence réseau artificielle — sont indispensables pour valider la résilience du système avant la mise en production. En 2026, des outils comme **LitmusChaos** avec des plugins GPU-aware permettent d'automatiser ces tests de résilience et de les intégrer dans les pipelines CI/CD. Pour approfondir, consultez Forensic Post-Hacking : Reconstruction et IA.

Checklist monitoring production LLM : (1) DCGM Exporter déployé sur chaque nœud GPU, (2) métriques vLLM/TGI exposées en Prometheus, (3) dashboard Grafana avec les 6 panneaux critiques, (4) alertes calibrées sur TTFT, taux de rejet, température GPU et erreurs ECC, (5) tracing OpenTelemetry sur les phases d'inférence, (6) runbooks d'incident validés par chaos engineering. Sans cette observabilité, le déploiement LLM en production est un vol à l'aveugle.

Ressources open source associées

GitHub KVortex — Optimisation KV-cache HF Model CyberSec-Assistant-3B-GGUF (quantifié)

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

Sources et références : ArXiv IA · Hugging Face Papers

FAQ

Qu'est-ce que Deployer des LLM en Production ?

Le concept de Deployer des LLM en Production est détaillé dans les premières sections de cet article, qui couvrent les fondamentaux, les enjeux et le contexte opérationnel. Pour un accompagnement sur ce sujet, contactez nos experts.

Pourquoi Deployer des LLM en Production est-il important en cybersécurité ?

La compréhension de Deployer des LLM en Production permet aux équipes de sécurité d'améliorer leur posture défensive. Les sections « Table des Matières » et « 1 Les Défis du Déploiement de LLM en Production » détaillent les raisons de cette importance. Pour un accompagnement sur ce sujet, contactez nos experts.

Comment mettre en œuvre les recommandations de cet article ?

Les recommandations pratiques sont détaillées tout au long de l'article, avec des commandes, des outils et des méthodologies éprouvées. La section « Conclusion » fournit une synthèse actionnable. Pour un accompagnement sur ce sujet, contactez nos experts.

Conclusion

Cet article a couvert les aspects essentiels de Table des Matières, 1 Les Défis du Déploiement de LLM en Production, 2 Architecture de Serving LLM. 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