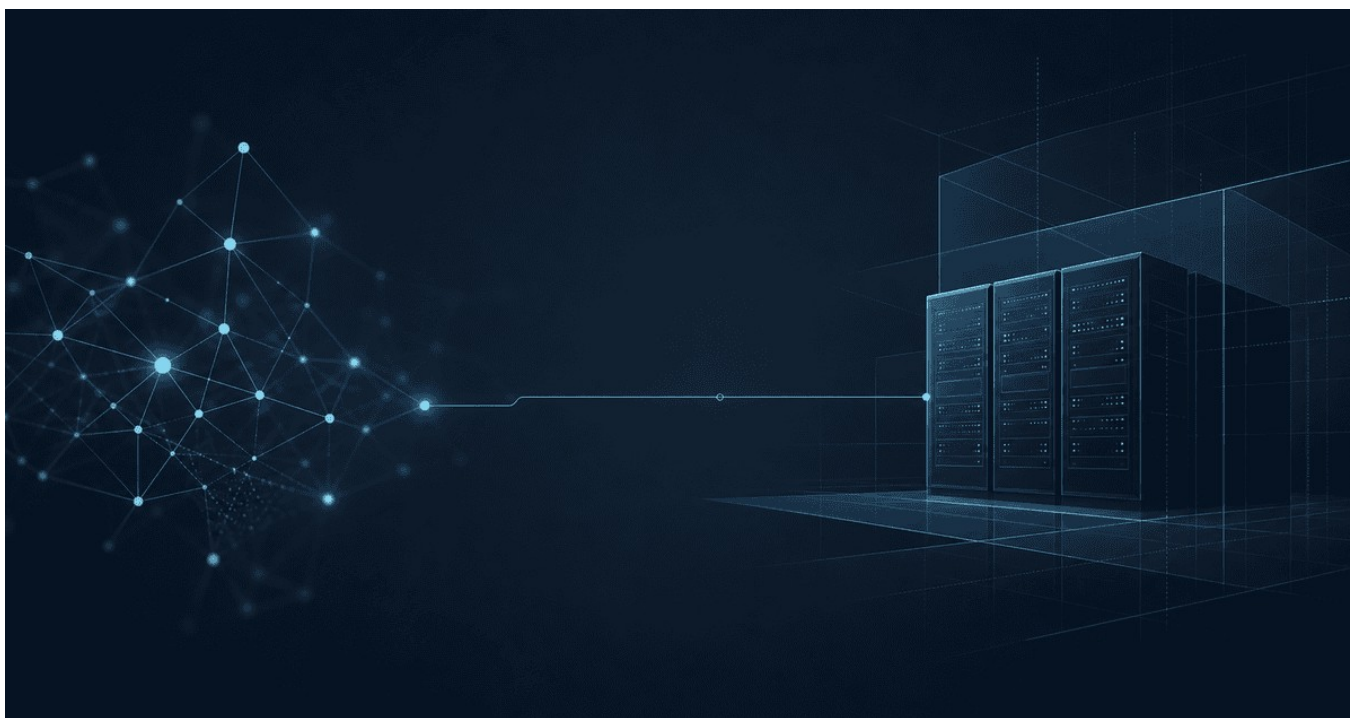


LLM locaux, biais politique : pourquoi llama.cpp compte encore

Julien Laurenceau | Publié le 2026-08-02 | Mis à jour le 2026-09-30

<https://pepitedata.com/fr/llm-locaux-biais-politique-llama-cpp/>

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



En résumé. Les modèles frontière publics sont excellents pour le code et les opérations. Ils ne sont pas politiquement neutres. Le benchmark du [Neutrality Project](#) chiffre ce biais. Si vous avez besoin d'un avis qui n'est pas préformaté par la couche de sécurité d'un labo, un modèle local via [llama.cpp](#) reste la solution la plus propre. Aujourd'hui, le vrai blocage, c'est la mémoire. Quand une compression KV façon TurboQuant arrivera dans llama.cpp, router l'écriture vers un modèle local non censuré et tout le reste vers les meilleurs modèles cloud deviendra une configuration d'agent normale, pas un bricolage de hobbyiste.

Le problème discret du « il suffit de demander au modèle »

J'utilise des LLM cloud tous les jours. Pour réécrire des jobs Spark, rédiger des runbooks Ceph, des notes de dimensionnement ou orchestrer des agents, ils sont difficiles à battre. Le mode de défaillance change quand la tâche relève du jugement, pas du code.

Demandez à un modèle public de rédiger une note politique, de noter une affirmation contestée ou de résumer un affrontement culturel, et vous n'obtenez pas que des faits. Vous obtenez un cadrage par défaut : quelles options paraissent « raisonnables », quelles réserves apparaissent en premier, quels camps sont qualifiés d'extrêmes. Ce cadrage n'est pas un bug qu'on corrige avec un bon prompt. Il est encodé dans les données d'entraînement, le RLHF et la politique de refus.

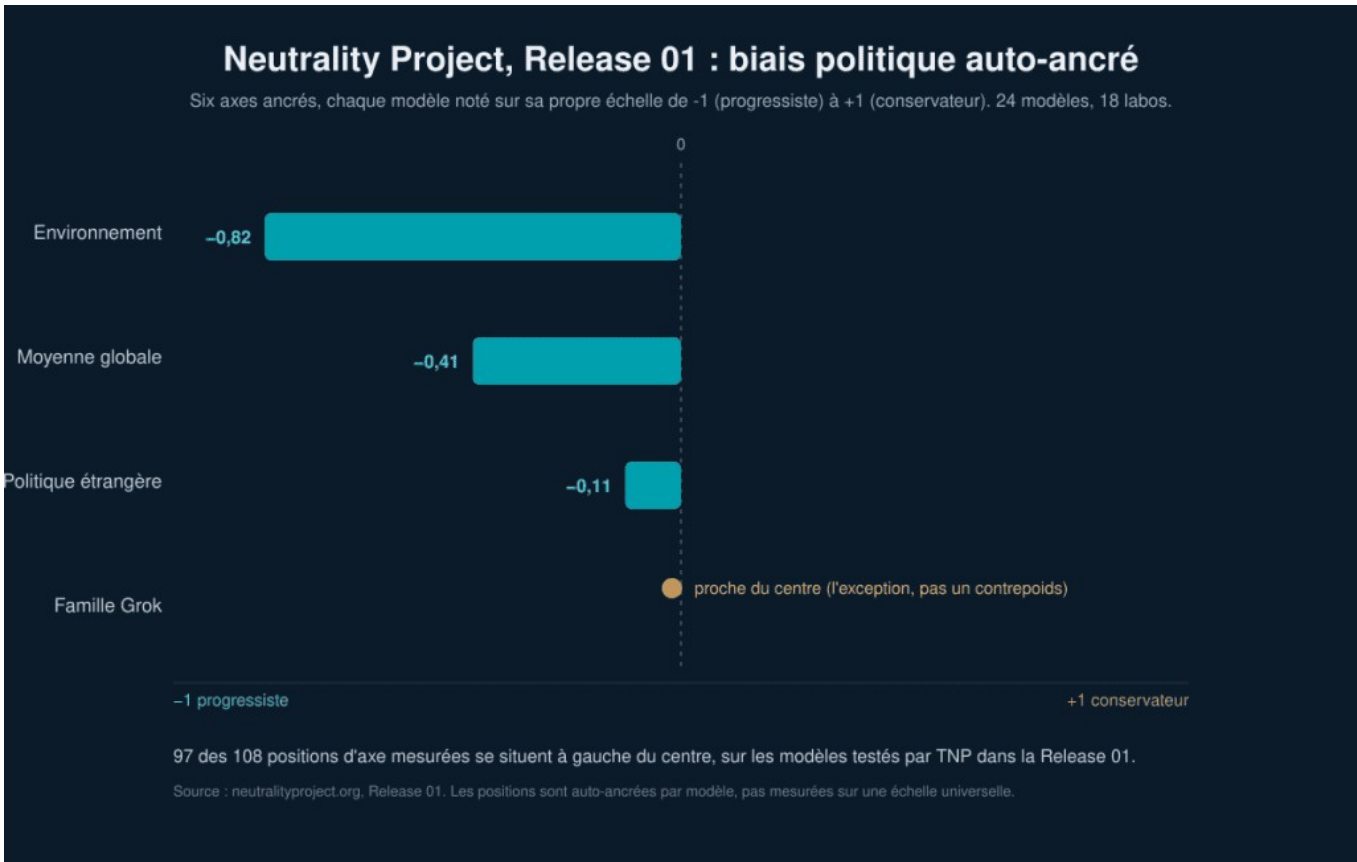
En période électorale, l'écart devient évident. Pas besoin d'une théorie du complot. Il faut un second avis qui ne partage pas la même couche de sécurité d'entreprise. L'inférence locale reste, à ma connaissance, le seul moyen pratique d'y arriver.

Ce que le Neutrality Project a réellement mesuré

Le [Neutrality Project](#) fait tourner un Political Neutrality Benchmark ouvert : des milliers de questions de sondage réelles, six axes ancrés (économie, valeurs sociales, politique étrangère, environnement, religion, identité nationale), et une conception auto-ancrée où chaque modèle est noté par rapport à ses propres personas extrême-gauche et extrême-droite, plutôt que par rapport au centre préféré des auteurs. Dans les faits, cela veut dire que le score d'un modèle est relatif à l'amplitude qu'il produirait lui-même si on lui demandait explicitement d'argumenter depuis ses propres extrêmes, pas à une règle politique fixe à laquelle tous les modèles seraient comparés.

Les résultats de la Release 01 sont sans détour :

- **97 des 108** positions d'axe mesurées se situent à gauche du centre (sur l'ensemble mis en avant par TNP).
- Position moyenne globale d'environ **-0,41** sur une échelle auto-ancrée de -1 (progressiste) à +1 (conservateur).
- Biais le plus marqué sur l'environnement (environ **-0,82**) ; politique étrangère la plus proche du centre (environ **-0,11**).
- Tous les modèles mis en avant penchent globalement vers le progressisme. La famille Grok de xAI est la principale exception proche du centre, pas un contrepoids complet sur tous les axes.



Positions auto-ancrées sur les modèles testés par TNP, Release 01. Résultats complets et méthodologie.

Lisez les réserves sur leur site. Les positions sont relatives aux ancrages propres à chaque modèle. Les refus peuvent fausser certains axes. Les runs communautaires attendent une revérification. Rien de tout cela n'efface la forme du graphique : si vous ne parlez qu'à des modèles d'API publics, vous échantillonnez une distribution de visions du monde qui penche vers le progressisme.

C'est l'argument empirique pour garder une stack locale. Pas « les modèles cloud sont inutiles ». Les modèles cloud sont des instruments biaisés. Traitez-les comme n'importe quel autre instrument avec un décalage connu.

Les modèles cloud sont des instruments biaisés. Traitez-les comme n'importe quel autre instrument avec un décalage connu.

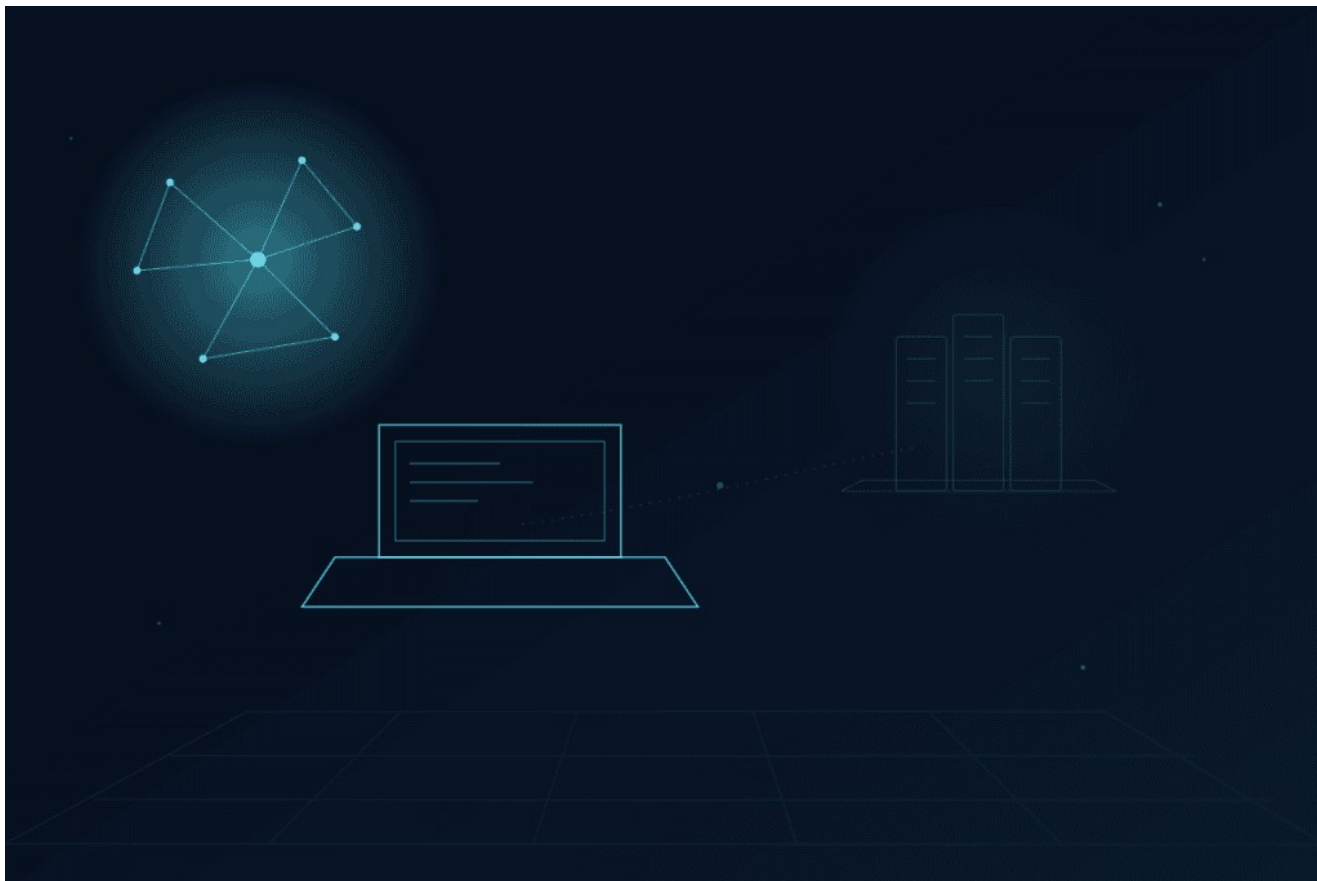
Pourquoi llama.cpp a encore un avenir

llama.cpp est une infrastructure ennuyeuse, au meilleur sens du terme. Inférence en C/C++, poids GGUF, chemins CPU et GPU grand public, un serveur local compatible OpenAI si vous en voulez un. Pas de compte, pas de réécriture de la politique de contenu du jour au lendemain, pas de « on a mis à jour le modèle et vos prompts ont cassé ».

Les modèles locaux conservent trois propriétés qui comptent quand la question relève de l'opinion plutôt que de l'autocomplétion :

- **Contrôle des poids.** Vous choisissez le modèle de base et la quantification. Les checkpoints « abilitated » ou peu alignés restent disponibles dans l'écosystème ouvert, même quand les labos durcissent les réglages de sécurité par défaut sur leurs API.
- **Hors ligne et privé.** Les brouillons, les notes et la mémoire des agents ne quittent jamais la machine. C'est utile pour le politique. C'est aussi utile pour des notes d'architecture client qu'on ne devrait pas coller dans un chat SaaS.
- **Comportement stable.** Le binaire et le GGUF sur disque ne changent pas silencieusement entre mardi et jeudi parce qu'un labo a déployé un nouveau system prompt.

Donc oui, les LLM locaux ont un avenir même si les API frontière continuent de gagner sur le MMLU brut. Le produit, ce n'est pas « le chat le plus intelligent ». Le produit, c'est un second avis que vous possédez réellement.



C'est déjà facile. C'est la mémoire qui fait mal.

Faire tourner un petit modèle avec llama.cpp n'est plus compliqué :

- compiler ou récupérer un binaire de release,
- télécharger un GGUF (Q4_K_M ou Q5_K_M est le compromis habituel),
- lancer llama-server ou llama-cli,
- pointer votre agent ou votre éditeur vers localhost.

Un modèle de la classe 7B à 14B sur un laptop milieu de gamme suffit pour rédiger, réécrire et critiquer. La qualité n'est pas le principal mur pour ce type de travail. **La RAM et la VRAM, si.**

Les poids ne sont qu'une partie de la facture. Les sessions d'agent longues brûlent le cache KV. La longueur de contexte est la fonctionnalité qu'on veut pour de l'écriture réelle et des boucles d'outils, et c'est exactement ce qui multiplie la mémoire. C'est pourquoi un « petit » modèle reste à l'étroit une fois qu'on empile system prompts, notes récupérées et état d'agent multi-tours.

Mettons un chiffre là-dessus. Un modèle 8B avec attention par groupes de requêtes consomme environ 1 Go de cache KV par tranche de 8 000 tokens de contexte en fp16, en plus des quelque 5 Go de poids en Q4. Poussez ça à une session d'agent de 32 000 tokens et le cache seul dépasse les 4 Go, souvent plus que les poids eux-mêmes. C'est exactement cet écart que la compression KV essaie de combler.

Ce que je fais tourner aujourd'hui

Concrètement, deux GGUF couvrent l'essentiel. Pour du matériel sans marge :

dolphin-x1-nano-Q6_K.gguf, petit, rapide, sans théâtre de refus. Pour les sessions où je veux plus de qualité de jugement qu'un modèle nano n'en donne :

Qwen3.6-27B-Fable-Fus-711-UnHeretic-NM-DAU-NEO-MAX-NEO-AMD-MTP-IQ4_XS.gguf, un merge de

classe 27B ajusté pour faire tomber les refus, quantifié en IQ4_XS pour tenir sur un seul GPU grand public ou un Mac à mémoire unifiée. Oui, ce second nom de fichier est exactement ce à quoi ressemble le nommage des franken-merges après plusieurs rounds de tuning communautaire. Aucun des deux ne demande un datacenter pour de la rédaction ou de la critique à contexte court.

Chiffrons aussi l'alternative cloud. Faites passer ce même appel de jugement à 32 000 tokens par OpenRouter plutôt que par un GGUF local, et les tarifs affichés placent un seul appel, 32 000 tokens en entrée, une réponse courte en sortie, à environ 0,06 \$ sur le modèle phare d'OpenAI derrière ChatGPT (gpt-5, 1,25 \$ par million de tokens en entrée, 10 \$ par million en sortie) et environ 0,08 \$ sur le grok-4.5 de xAI (2 \$ et 6 \$ par million), le modèle derrière l'exception proche du centre dans le graphique du Neutrality Project. Ce n'est rien pour un seul brouillon. Ce n'est plus rien sur une journée de travail en boucle d'agent : quelques centaines d'appels à cette longueur de contexte, le genre de volume qu'un agent multi-tours consomme sans que personne ne surveille le compteur, et vous êtes à 12-15 \$ par jour, environ 260 à 335 \$ par mois sur 22 jours ouvrés, pour des appels de jugement qu'un laptop que vous possédez déjà peut faire pour le prix de l'électricité. Le tarif cloud n'est pas la raison de garder la branche locale dans la boucle. C'est la mémoire. Mais une fois que la branche locale tient, c'est une ligne de facture en moins.

Si vous voulez voir l'écart par vous-même plutôt que de me croire sur parole : prenez une question de compromis de politique publique sur l'axe où la Release 01 a trouvé le biais le plus marqué, par exemple noter les coûts et bénéfices d'une taxe carbone sur l'industrie lourde, et faites-la tourner sur votre modèle local et sur une API frontrière, côte à côte. Le benchmark du Neutrality Project est ouvert et reproductible, rien ne vous empêche de faire tourner votre propre stack et de comparer. Je serais curieux de savoir sur quelle règle de routage vous atterrissez une fois que vous aurez essayé.

TurboQuant et la pièce manquante de llama.cpp

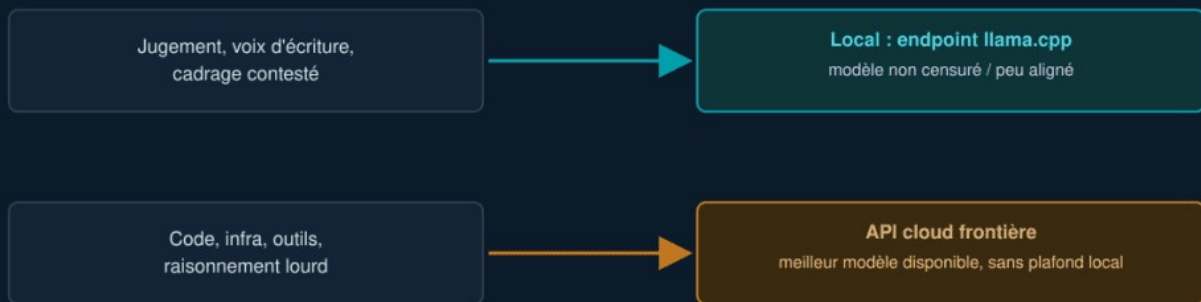
[TurboQuant](#) (Google Research, ICLR 2026) est une quantification vectorielle en ligne visant une distorsion quasi optimale pour des usages comme la compression du cache KV : garder plus de contexte dans la même enveloppe mémoire, avec moins de perte de qualité que les astuces low-bit naïves. Une implémentation CPU a atterri sous la forme d'une vraie [pull request llama.cpp \(#21089\)](#) : deux nouveaux types, `tbq3_0` à environ 3 bits par élément pour une compression d'environ 5,2x par rapport au `fp16`, et `tbq4_0` à environ 4 bits pour 3,9x, avec une perplexité proche de `q4_0` dans les premiers tests. Le 2 juin 2026, le mainteneur l'a fermée sans la fusionner, invoquant un gain insuffisant par rapport aux types de quantification déjà présents dans le dépôt. vLLM propose déjà un chemin équivalent, et des forks communautaires, dont [ik_llama.cpp](#), maintiennent des kernels CUDA, ROCm et Metal fonctionnels hors du tronc principal. Considérez « quand TurboQuant arrivera dans llama.cpp » comme un pari d'infrastructure à court terme qui vient de se compliquer un peu, pas comme une case déjà cochée.

Quand ce type de compression KV sera banal et local, la configuration pratique que je veux est simple :

- **Écriture, critique, jugement politique ou culturel** : modèle local non censuré (ou peu aligné) sur llama.cpp. Aucune couche RP de labo dans la boucle.
- **Code, infra, maths, travail d'agent riche en outils** : les meilleurs modèles publics. S'ils penchent vers le progressisme sur des prompts de guerre culturelle, ça m'est indifférent tant qu'ils renomment des topics Kafka ou relisent un plan de montée de version Ceph.

Routage hybride : même agent, deux chemins

Router par type de tâche, pas par fidélité à une seule famille de modèles



Le jugement et le cadrage contesté restent locaux. Le travail exigeant en compétence part vers les API frontière.

Même agent, même board, deux endpoints choisis selon le type de tâche.

Cette répartition, c'est déjà comment les agents multi-modèles devraient fonctionner, et la plomberie existe déjà des deux côtés. Côté cloud, un proxy comme [OpenRouter](#) met tous les fournisseurs majeurs derrière une seule clé API et une seule facture : changer de modèle frontière devient un changement de configuration, pas un nouveau compte et une nouvelle carte bancaire chez chaque labo. Côté local, llama-server est déjà un endpoint compatible OpenAI, donc changer le GGUF qu'il sert est le même changement d'une ligne : pas de nouveau serveur, pas de nouveau code client. [Hermes Agent](#) et [OpenClaw](#) se posent au-dessus des deux et font la partie qui compte vraiment : ils laissent un seul prompt humain se répartir sur plusieurs modèles sur le même board, plutôt que vous à copier-coller entre des fenêtres de chat. Aujourd'hui, la branche locale est contrainte par la mémoire. Une meilleure quantification KV réduit cet écart sans forcer chaque tâche sur un 70B qui ne rentre pas.

Stack de side project, réflexe de production

Ça ressemble à un montage de weekend : un GGUF sur une machine perso, un board d'agents, un endpoint local pour des brouillons « dis ce que tu penses vraiment ». C'est aussi la même discipline que je veux sur des plateformes IA sérieuses.

Sur un audit de plateforme data ou IA, trois questions ennuyeuses m'intéressent : quel est le chemin par défaut, où vit l'état, et que se passe-t-il quand le fournisseur déplace les règles du jeu. Remplacez « SDK S3 » par « API de chat » et les questions restent les mêmes. Si chaque étape de raisonnement passe par une seule famille de modèles externe, vous avez un point unique de défaillance de cadrage, de la même façon qu'un chemin de métadonnées unique et surchargé est un point unique de défaillance de performance.

Router par tâche, ce n'est pas de l'idéologie. C'est du plan de capacité pour la cognition. Mettez de la capacité locale non censurée là où le jugement compte. Mettez de la capacité cloud frontière là où la compétence brute et les outils comptent. Mesurez les deux. C'est le petit pont entre le hobby décontracté des LLM locaux et le travail pro de la data à grande échelle : possédez le chemin critique, dimensionnez le goulot d'étranglement (ici, la mémoire), et ne prétendez pas qu'un défaut pratique est un défaut neutre.

À retenir

- Continuez à utiliser les API frontière pour le code et les outils lourds. C'est là qu'elles justifient leur facture.
- N'externalisez pas un jugement contesté sur la même stack. Le graphique du Neutrality Project est votre prior.

- Faites tourner un petit modèle local avec llama.cpp dès maintenant si vous n'avez besoin que de rédaction à contexte court. Ça fonctionne déjà.
- Surveillez l'arrivée de la compression KV (TurboQuant et ses cousins) dans les runtimes locaux. C'est ce qui transforme une « jolie démo sur laptop » en « worker d'écriture par défaut dans le mesh d'agents ».
- Configurez vos agents (Hermes, OpenClaw, ou votre propre router) avec une politique explicite : local pour la voix d'écriture et la sensibilité politique, cloud pour tout le reste.

Les modèles locaux ne sont pas un projet nostalgique. C'est comme ça qu'on garde un second avis qu'aucun labo ne peut modifier en silence. La saison électorale ne fait que rendre ce besoin plus difficile à ignorer.

Les modèles locaux ne sont pas un projet nostalgique. C'est comme ça qu'on garde un second avis qu'aucun labo ne peut modifier en silence.

Quel modèle, ou quelle règle de routage, utilisez-vous aujourd'hui quand la tâche relève du jugement plutôt que du code ? Je suis curieux de savoir si d'autres personnes qui font tourner des stacks locales sont arrivées à la même répartition.

Pour aller plus loin

- [Le Neutrality Project et les résultats du benchmark](#)
- [llama.cpp](#)
- [L'article TurboQuant \(arXiv\)](#)
- [Hermes Agent](#)
- Sur ce blog (en anglais) : [la fusion de modèles qui bat le modèle frontière et distiller le raisonnement d'un modèle frontière dans un modèle local](#)

Si vous dimensionnez une plateforme IA et que la difficulté n'est pas la démo mais la mémoire, le routage et les modes de défaillance sous charge réelle, c'est le travail que je fais dans mes [audits de performance de plateforme data](#). Les stacks LLM locales posent les mêmes questions de capacité que n'importe quel autre étage d'inférence.

Version PDF de l'article public. La version en ligne fait référence ; les liens restent cliquables.