# 130K Tokens de Contexte sur une RTX 3090

Optimisation LLM • Quantification KV Cache • Architecture Client-Serveur

## Résultats

| | |
|---|---|
| **130 000**<br>tokens de contexte | **−50%**<br>utilisation VRAM |
| **+179%**<br>débit (23 → 64 t/s) | **Dual-LLM**<br>14B + 3B simultanément |

**Matériel :** NVIDIA RTX 3090 (24 Go VRAM) — matériel grand public, résultats professionnels.

## Le Défi

Mon système d'IA cognitive **SecondMind** repose sur une architecture à double LLM : un modèle principal (Qwen2.5-14B) pour le raisonnement, et un modèle juge (Qwen2.5-3B) pour la validation. L'objectif : étendre la fenêtre de contexte de 32K à 128K tokens pour l'analyse de documents longs.

### Le problème mathématique

Un contexte de 128K tokens avec un cache KV en FP16 requiert théoriquement **36,2 Go de VRAM** — impossible sur une carte de 24 Go. La solution évidente : la quantification du cache KV, qui promet une réduction de 50 à 75% de l'empreinte mémoire.

*Le problème : ça ne fonctionnait pas.*

## Investigation

Une approche systématique a révélé trois problèmes critiques :

- **Bindings CUDA corrompus** — Le wheel pré-compilé de llama-cpp-python était compilé sans support CUDA, malgré son étiquetage. Diagnostic : *llama_print_system_info()* ne mentionnait aucun device CUDA.
- **DLLs manquantes** — Les fichiers *ggml-cuda.dll*, *cublas64_12.dll* étaient absents du paquet installé.
- **Incompatibilité Flash Attention** — La quantification KV nécessite Flash Attention, qui ne compile pas correctement sur Windows avec les bindings Python. Impasse architecturale.

## Solution : Pivot Architectural

Plutôt que de lutter contre les limitations des bindings Python, pivot vers le **serveur natif llama.cpp** (CLI) avec communication HTTP.

### Pourquoi ce choix

- Builds Windows CUDA officiels, pré-compilés et testés
- Flash Attention et quantification KV fonctionnels out-of-the-box
- Meilleure isolation : un crash serveur ne tue pas le processus principal
- Surcoût HTTP négligeable (~1-2ms vs ~15ms/token d'inférence)

## Benchmarking et Optimisation

Trois configurations ont été testées systématiquement pour identifier le compromis optimal entre capacité de contexte, utilisation VRAM et débit.

### Méthodologie de vérification

Pour chaque configuration, un protocole de validation en 4 étapes :

1. **Validation des logs serveur** — Vérification des messages *llama_kv_cache: CUDA0 KV buffer size* pour confirmer la quantification effective.
2. **Monitoring GPU** — Mesure de l'utilisation VRAM via *nvidia-smi* en temps réel.
3. **Test de performance** — Génération de 1000 tokens avec un prompt constant pour mesurer le débit.
4. **Vérification qualité** — Validation de la cohérence sur des conversations multi-tours pour détecter toute dégradation.

### Résultats détaillés

| Config | Contexte | Cache | VRAM Cache | VRAM Total | Débit | Temps 1K |
|---|---|---|---|---|---|---|
| Baseline | 32K | FP16 | 6 144 Mo | 14,5 Go | 23 t/s | 43,5 s |
| Config A | 32K | Q8_0 | 3 264 Mo | 10,6 Go | 63,9 t/s | 15,6 s |
| Config B | 64K | Q8_0 | 6 528 Mo | 21,4 Go | 65,8 t/s | 15,2 s |
| **Config C** | **128K** | **Q4_0** | **6 912 Mo** | **21,9 Go** | **64,2 t/s** | **15,6 s** |

### Observations clés

**1. Économies de VRAM (Config A)**

- 47% de réduction du cache (6 144 → 3 264 Mo)
- 27% de réduction totale de la VRAM (14,5 → 10,6 Go)

**2. Amélioration contre-intuitive de la performance**

- Augmentation du débit de 2,7x (23 → 64 t/s)
- *Cause :* Meilleure efficacité des kernels CUDA avec les opérations quantifiées

**3. Le compromis idéal (Config B)**

- 2x extension du contexte (32K → 64K)
- Maintient un débit élevé (65,8 t/s)
- Marge VRAM confortable (2,6 Go libres)

**4. Configuration maximale (Config C — retenue)**

- 4x extension du contexte (32K → 128K)
- Même VRAM que Config B (la quantification Q4 compense le contexte 2x plus grand)
- **Aucune dégradation de qualité** détectée lors des tests multi-tours

# Impact Réel

| Métrique | Baseline | Optimisé | Gain |
|---|---|---|---|
| Contexte Max | 32 768 | 131 072 | **+300%** |
| VRAM (128K théorique) | 36,2 Go | 21,9 Go | **−39%** |
| Débit | 23 t/s | 64,2 t/s | **+179%** |
| Efficacité VRAM | 2 257 tok/Go | 5 983 tok/Go | **+165%** |

## Cas d'usage débloqués

**Analyse de documents longs**

- Articles de recherche entiers (20-30K tokens)
- Contrats juridiques (50-80K tokens)
- Bases de code complètes (100K+ tokens)

**Conversations étendues**

- Plus de 400 échanges avec contexte complet
- Fin de la troncature de la fenêtre de contexte

**Flux de travail Dual-LLM**

- LLM Principal (14B) + LLM Juge (3B) simultanément
- Total : 21,9 + 4,2 = ~26 Go (multi-GPU ou séquentiel)

# Principes

*« Les wheels pré-compilés mentent. »*

Toujours vérifier le support CUDA avec les outils de diagnostic. Les étiquettes ne garantissent rien.

*« Flexibilité architecturale > optimisation monolithique. »*

Le pivot vers une architecture client-serveur a résolu des problèmes insolubles en approche intégrée. Le surcoût HTTP (<1%) est négligeable.

*« Mesurer, ne pas supposer. »*

Chaque configuration a été benchmarkée systématiquement. Les résultats ont contredit les "évidences" : le débit a augmenté là où on attendait une dégradation.

*« N'acceptez pas les limitations comme une fatalité. »*

"128K ne tiendra pas sur 24 Go" — prouvé faux. "23 t/s est le maximum" — atteint 64 t/s. L'investigation technique et la volonté de remettre en question ouvrent des possibilités.

—

Maxime Gagné • Architecte Cognitif • Novembre 2025