LA MÉMOIRE DES MACHINES

Du KV-Cache au Context Engineering

PATRICE HUETZ

La Mémoire des Machines

Du KV-Cache au Context Engineering

Patrice Huetz

patrice-huetz.fr

patrice-huetz.fr · contact@patrice-huetz.fr

Avant-propos

Il existe un paradoxe que tout développeur travaillant avec des modèles de langage finit par rencontrer. Les LLM sont, par construction, des machines à oublier. Ils ingèrent des centaines de milliers de tokens, semblent tout comprendre — puis, au détour d'un échange un peu long, réintroduisent un bug qu'on venait de corriger, oublient une décision architecturale prise vingt messages plus tôt, ou confondent deux fichiers pourtant distincts. Ce n'est pas un défaut d'intelligence. C'est un défaut de mémoire.

Ce livre est né de la collision de deux projets.

Le premier, **TurboQuant**, est un moteur d'inférence écrit en Rust. Son objectif : compresser le KV-cache — cette structure de données colossale qui stocke l'état attentionnel du modèle — pour qu'il tienne dans la mémoire GPU sans sacrifier la qualité des réponses. Quantification mixte, allocation dynamique par couche, éviction intelligente des tokens peu informatifs. Du travail au niveau du silicium, où chaque octet compte.

Le second, **Code Buddy**, est un assistant de développement en TypeScript. Son objectif : gérer le contexte applicatif — décider quels fichiers injecter dans le prompt, quand résumer l'historique, comment structurer les instructions pour que le modèle ne perde pas le fil d'un projet de 200 000 lignes de code. Du travail au niveau du prompt, où chaque token compte.

Ces deux projets attaquent le même problème fondamental — *la mémoire finie des LLM* — mais depuis des couches radicalement différentes de la stack. Et c'est précisément cette dualité qui manque dans la littérature actuelle. Les articles sur le KV-cache parlent rarement de context engineering. Les guides sur le prompt design ignorent les contraintes matérielles qui dictent les fenêtres de contexte. Or, pour construire des systèmes d'IA réellement fiables, il faut comprendre les deux.

La Mémoire des Machines couvre donc l'intégralité de la chaîne, du silicium au prompt :

- **Partie I** — La mémoire telle qu'elle fonctionne : attention, KV-cache, fenêtres de contexte, dégradation mesurable.
- **Partie II** — Les techniques de compression : quantification, éviction, architectures alternatives (MLA, GQA, sliding window).
- **Partie III** — Le context engineering applicatif : compaction, mémoire structurée, orchestration d'agents, prévention plutôt que compression.
- **Partie IV** — La convergence : comment les deux couches interagissent, et ce que cela implique pour l'architecture des systèmes de demain.

Pour qui est ce livre

Ce livre s'adresse à trois publics :

- **Les développeurs IA** qui construisent des applications sur des LLM et veulent comprendre pourquoi leur agent « oublie » après 30 minutes de conversation — et comment l'empêcher.
- **Les ingénieurs ML** qui optimisent l'inférence et cherchent un panorama structuré des techniques de compression du KV-cache, avec benchmarks et arbitrages.
- **Les architectes d'agents** qui conçoivent des systèmes multi-agents et doivent prendre des décisions éclairées sur la gestion de la mémoire à chaque couche.

Aucun prérequis en recherche ML n'est nécessaire, mais une familiarité avec Python et les concepts de base du deep learning (tenseurs, couches, fonctions d'activation) est supposée. Les passages mathématiques sont toujours accompagnés d'intuitions et de visualisations.

Chaque chapitre commence par une situation concrète — un bug réel, une métrique qui décroche, un agent qui déraille — avant d'expliquer le concept sous-jacent et de proposer des solutions pratiques. Parce que la mémoire des machines n'est pas un problème théorique. C'est le problème que vous rencontrez chaque jour, dès que votre conversation dépasse quelques milliers de tokens.

Patrice Huetz, avril 2026

CHAPITRE 1

Pourquoi les LLM Oublient

La conversation qui déraille

Vous êtes en train de refactorer un module d'authentification avec votre assistant IA. Les quarante premiers messages se passent admirablement : le modèle comprend l'architecture, respecte vos conventions de nommage, propose des abstractions pertinentes. Puis, au message 47, il réintroduit une dépendance circulaire que vous aviez explicitement éliminée au message 12. Au message 53, il oublie que vous avez migré de JWT vers des sessions côté serveur. Au message 58, il mélange deux fichiers et produit un code qui ne compile plus.

Ce n'est pas un modèle défectueux. C'est un modèle qui oublie. Et pour comprendre pourquoi, il faut remonter au mécanisme fondamental qui lui permet de « comprendre » : l'attention.

L'attention transformer : tout est relation

Un LLM ne « lit » pas un texte comme un humain. Il le transforme en une séquence de vecteurs numériques — les *embeddings* — puis calcule, pour chaque token, à quel point il doit « faire attention » à chaque autre token de la séquence. C'est le mécanisme d'attention, introduit par Vaswani et al. en 2017 dans l'article fondateur *Attention Is All You Need*.

Voici l'intuition. Considérez la phrase :

> *« Le serveur a renvoyé une erreur parce qu'il n'avait plus de mémoire. »*

Quand le modèle traite le mot « il », il doit déterminer que « il » fait référence à « serveur » et non à « erreur » ou à « mémoire ». Pour cela, chaque token génère trois vecteurs :

- **Query (Q)** : « Que cherche ce token ? » — « il » cherche son antécédent.
- **Key (K)** : « Que propose ce token ? » — « serveur » propose qu'il est un sujet valide.
- **Value (V)** : « Quelle information ce token transporte-t-il ? » — le contenu sémantique de « serveur ».

Le score d'attention entre deux tokens est le produit scalaire de leurs vecteurs Q et K, normalisé par la racine carrée de la dimension :

```
Attention(Q, K, V) = softmax(QKᵀ / √d_k) · V
```

Le **softmax** convertit les scores bruts en probabilités : des poids entre 0 et 1 qui somment à 1. C'est cette normalisation qui est à la fois la force et la faiblesse du mécanisme.

Les têtes d'attention multiples

En pratique, un transformer n'utilise pas une seule fonction d'attention mais plusieurs en parallèle — les *attention heads*. Claude Opus utilise 128 têtes par couche sur 80 couches. Chaque tête apprend à capturer un type de relation différent : syntaxique, sémantique, positionnelle, logique. Les résultats sont concaténés puis projetés.

Cette architecture permet au modèle de maintenir simultanément de nombreuses « perspectives » sur le texte. Mais chaque tête partage la même contrainte : le softmax force une distribution de probabilités sur *tous* les tokens de la séquence.

Les tokens : l'unité de base

Avant d'aller plus loin, clarifions une notion fondamentale. Un LLM ne travaille pas avec des mots, mais avec des **tokens** — des fragments de texte découpés par un algorithme de tokenisation (BPE, SentencePiece, ou similaire).

En anglais, un token correspond en moyenne à **0,75 mot**. Le mot « understanding » est un seul token, mais « misunderstanding » en fait deux (« mis » + « understanding »).

En français, le ratio tombe à environ **0,5 mot par token**. Les accents, les conjugaisons complexes et les mots composés fragmentent davantage le texte. Le mot « développement » consomme typiquement 3 tokens (« dé » + « veloppe » + « ment »). Le code source, avec ses symboles et sa syntaxe, est encore plus gourmand.

Conséquence directe : une fenêtre de 200 000 tokens ne représente pas 200 000 mots. En français technique mêlé de code, c'est plutôt **80 000 à 100 000 mots** — soit un roman de taille moyenne, pas une encyclopédie.

Langue / contenu	Ratio mots/token	200K tokens ≈
Anglais courant	0,75	150 000 mots
Français courant	0,50	100 000 mots
Code Python	0,35	70 000 mots-équivalents
JSON / YAML	0,25	50 000 mots-équivalents
Français + code mélangé	0,45	90 000 mots

La fenêtre de contexte : pas une limite de taille, une limite de précision

C'est ici que la confusion règne. Quand un fournisseur annonce « 200K tokens de contexte », la plupart des développeurs comprennent : « Je peux mettre 200 000 tokens dans le prompt et le modèle les traitera tous. » C'est techniquement vrai. Le modèle *acceptera* ces tokens. Mais les traitera-t-il avec la même qualité ? Non.

Le softmax : tout est relatif

Reprenons la formule d'attention. Le softmax produit une distribution de probabilités sur l'ensemble de la séquence. Si la séquence fait 1 000 tokens, chaque token reçoit en moyenne un poids de 1/1 000. Si elle fait 200 000 tokens, ce poids moyen tombe à 1/200 000 — soit 200 fois moins.

Bien sûr, le softmax n'est pas uniforme : il concentre les poids sur les tokens les plus pertinents. Mais même avec cette concentration, l'ajout de tokens non pertinents dans la séquence « dilue » l'attention disponible. C'est comme chercher un mot dans un dictionnaire de poche versus dans un dictionnaire encyclopédique : l'information est là, mais le bruit environnant ralentit et dégrade la recherche.

Ce phénomène porte un nom : la **dilution attentionnelle**. Plus la séquence est longue, plus il est difficile pour le modèle de concentrer son attention sur les tokens qui comptent vraiment.

L'encodage positionnel : le sens de l'ordre

Les transformers n'ont pas de notion intrinsèque de position. Sans encodage positionnel, la phrase « le chat mange la souris » et « la souris mange le chat » produiraient la même représentation. Pour résoudre ce problème, on ajoute à chaque token une information de position.

Les modèles modernes utilisent des encodages positionnels rotatifs (**RoPE** — Rotary Position Embedding), qui encodent la position relative entre deux tokens plutôt que leur position absolue. Cela permet d'extrapoler au-delà de la longueur d'entraînement — en théorie.

En pratique, RoPE souffre d'une dégradation progressive pour les positions éloignées. Au-delà d'un certain seuil (qui dépend de la base de fréquence et de la longueur d'entraînement), les relations positionnelles deviennent bruitées. Le modèle « sait » qu'un token est « quelque part au début », mais perd la granularité fine.

Fenêtres annoncées vs fenêtres effectives

Si les fenêtres de contexte étaient parfaites, ce livre n'existerait pas. Mais elles ne le sont pas, et les données sont sans appel.

Le rapport Chroma (2025) : 100 % des modèles dégradent

En mars 2025, l'équipe de Chroma a publié un rapport dévastateur intitulé *Context is Not a Crutch*. Leur méthodologie : tester la capacité de 18 modèles de langage à retrouver des informations précises (des « aiguilles ») dans des contextes de taille croissante, en mesurant la précision à intervalles réguliers.

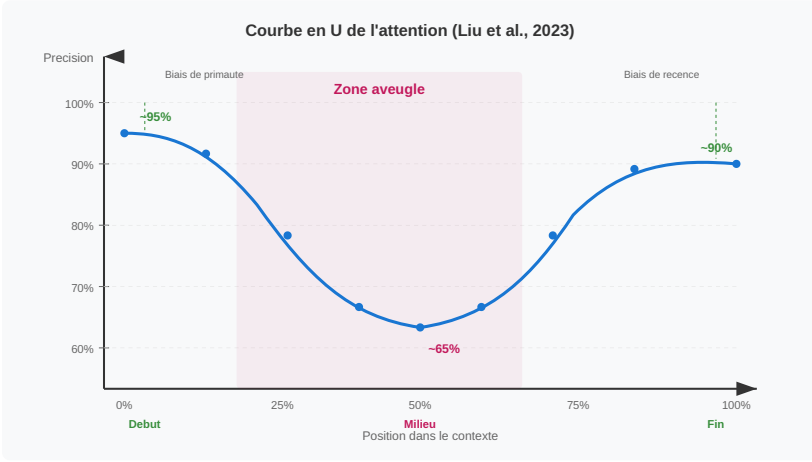
Résultat : **100 % des modèles testés montrent une dégradation mesurable bien avant d'atteindre leur limite de fenêtre annoncée.** Aucune exception. Ni Claude, ni GPT-4, ni Gemini, ni les modèles open source.

La dégradation n'est pas brutale — ce n'est pas une falaise. C'est une pente douce mais continue, qui commence parfois dès 20 à 30 % de remplissage de la fenêtre.

La courbe en U : Lost in the Middle

En 2023, Liu et al. (Stanford) ont publié un article fondateur : *Lost in the Middle: How Language Models Use Long Contexts*. Leur découverte : les LLM traitent significativement mieux les informations situées au **début** et à la **fin** du contexte, et significativement moins bien celles situées au **milieu**.

La courbe de performance en fonction de la position dans le contexte forme un U caractéristique :



Courbe en U de l'attention — les LLM traitent mieux le début et la fin du contexte
L'explication est double :

1. **Biais de primauté** : les premiers tokens bénéficient d'un traitement privilégié car ils servent de « contexte fondateur » pour tous les tokens suivants.
2. **Biais de récence** : les derniers tokens sont les plus frais dans la mémoire de travail du modèle (les mécanismes d'attention causale leur donnent accès à tous les tokens précédents sans compétition).

Les tokens du milieu, eux, sont pris en tenaille : trop loin du début pour bénéficier du biais de primauté, trop loin de la fin pour bénéficier de la récence. Ils entrent dans ce que nous appellerons la **zone morte** du contexte.

Au-delà de 50 % : le biais de récence domine

Veseli et al. (2025) ont affiné cette analyse en montrant qu'au-delà de 50 % de remplissage de la fenêtre de contexte, le biais de récence devient dominant. Autrement dit : plus le contexte est long, plus le modèle « oublie » activement le début au profit de la fin.

Cela a des implications pratiques majeures. Si vous construisez un agent de coding qui accumule les messages au fil d'une session de 2 heures, les instructions initiales (conventions de nommage, architecture cible, contraintes de sécurité) seront progressivement noyées par les échanges récents (corrections de bugs, discussions tangentielles, logs d'erreur).

Panorama des fenêtres de contexte (avril 2026)

Modèle	Fenêtre annoncée	Fenêtre effective estimée	Notes
Claude Opus 4	200K tokens	~120K tokens	Dégradation progressive au-delà de 100K
Claude Sonnet 4	200K tokens	~100K tokens	Meilleur rapport coût/qualité en contexte moyen
GPT-4o	128K tokens	~80K tokens	Dégradation marquée au-delà de 64K
GPT-4.1	1M tokens	~200K tokens	Fenêtre étendue, précision variable
Gemini 2.5 Pro	1M tokens	~300K tokens	Meilleure rétention longue distance
Gemini 2.5 Flash	1M tokens	~200K tokens	Compromis vitesse/rétention
Llama 4 Scout	10M tokens	~500K tokens	MoE, 16 experts, contexte théorique
Mistral Large	128K tokens	~60K tokens	Bonne performance en contexte court
DeepSeek-V3	128K tokens	~64K tokens	MLA architecture, KV-cache compressé
Qwen 3	128K tokens	~80K tokens	Bon ratio multilingue

> ***Note** : les « fenêtres effectives » sont des estimations basées sur les benchmarks publics de type needle-in-a-haystack et les rapports de dégradation (Chroma, NoLiMa). Elles varient selon la tâche, la langue et le type de contenu.*

L'écart entre « annoncé » et « effectif » est systématique. Ce n'est pas du marketing mensonger — les modèles *acceptent* bien ces longueurs de contexte. Mais accepter n'est pas comprendre, et comprendre n'est pas retenir.

L'analogie de la mémoire de travail

Pour rendre cette réalité intuitive, pensons à la mémoire de travail humaine. Un adulte moyen peut maintenir environ 7 ± 2 éléments dans sa mémoire de travail (le fameux « nombre magique » de Miller, 1956). Cela ne signifie pas qu'il ne peut pas lire un document de 100 pages — mais qu'à tout instant, seuls 5 à 9 éléments sont activement manipulables.

Un LLM est dans une situation analogue, à une échelle différente. Sa « mémoire de travail » — la capacité du mécanisme d'attention à maintenir des relations précises entre tokens — est bien inférieure à sa fenêtre de contexte nominale. Il peut *stocker* 200 000 tokens, mais il ne peut en *manipuler activement* qu'une fraction.

La différence cruciale : un humain sait qu'il oublie. Il prend des notes, relit des passages, structure son travail en conséquence. Un LLM, lui, ne sait pas qu'il oublie. Il produit ses réponses avec la même confiance apparente, que le contexte fasse 1 000 ou 200 000 tokens. C'est ce qui rend le problème insidieux : la dégradation est silencieuse.

Ce que cela implique

Si vous retenez une seule chose de ce chapitre, que ce soit ceci : **la fenêtre de contexte n'est pas un seau qu'on remplit.** C'est un projecteur dont le faisceau s'élargit et s'affaiblit à mesure qu'on augmente la portée. Mettre plus de tokens dans le contexte, c'est élargir le faisceau. Au-delà d'un certain point, la lumière ne suffit plus à éclairer quoi que ce soit avec précision.

Ce constat pose la question centrale de ce livre : si les LLM oublient inévitablement, comment concevoir des systèmes qui compensent cet oubli ? La réponse se situe à deux niveaux — la compression matérielle (Partie II) et l'ingénierie du contexte (Partie III). Mais avant d'explorer les solutions, il faut comprendre la profondeur du problème. C'est l'objet du chapitre suivant.

Points clés

- Le mécanisme d'attention (Q, K, V + softmax) est puissant mais fondamentalement limité par la normalisation softmax, qui dilue les poids à mesure que la séquence s'allonge.
- Un token ≠ un mot. En français technique mêlé de code, 200K tokens ≈ 90 000 mots.
- 100 % des modèles testés (Chroma 2025) montrent une dégradation avant leur limite de fenêtre annoncée.
- La courbe en U (Lost in the Middle) : le début et la fin du contexte sont bien traités ; le milieu est une zone morte.
- Au-delà de 50 % de remplissage, le biais de récence domine (Veseli et al. 2025).
- La fenêtre effective d'un modèle est systématiquement inférieure à sa fenêtre annoncée — souvent de 40 à 60 %.
- La dégradation est silencieuse : le modèle ne signale pas qu'il oublie, il continue avec la même confiance apparente.