Tokens et chunks dans les IA génératives

Guide pratique pour optimiser vos usages

Stéphane FOSSE

fosse.fr

24 février 2026

Résumé

Un token n'est pas un mot. Un chunk n'est pas un paragraphe. Ces deux notions structurent l'intégralité de ce que les grands modèles de langage peuvent lire, comprendre et produire. Ce guide les explore en profondeur, avec un regard centré sur les implications pratiques : pourquoi une traduction en français coûte plus cher qu'en anglais, pourquoi un chatbot oublie ce que vous lui avez dit deux heures plus tôt, et pourquoi un système de recherche documentaire peut retourner des réponses absurdes sur un contrat parfaitement rédigé.

Table des matières

1 Le token : l'atome du langage machine

Quand vous tapez une phrase dans ChatGPT, Claude ou Gemini, le modèle ne voit pas ce que vous voyez. Il ne lit ni mots ni caractères. Il reçoit une séquence de nombres entiers, chacun représentant un fragment de texte appelé **token**. Ce découpage est le premier geste, invisible, que fait n'importe quel grand modèle de langage (LLM, pour Large Language Model) avant même de commencer à traiter votre requête.

La notion de token est née d'une contrainte technique fondamentale. Les modèles de type Transformer — l'architecture décrite par Vaswani et ses collègues de Google Brain en 2017 dans l'article *Attention Is All You Need* [14] — traitent le texte comme une séquence d'éléments discrets. Chaque élément est converti en vecteur numérique, puis ces vecteurs sont mis en relation via le mécanisme d'attention. Ce mécanisme calcule, pour chaque position dans la séquence, à quel point elle doit « prêter attention » à toutes les autres. Sa complexité croît de façon quadratique avec le nombre de tokens : doubler la longueur d'un texte ne double pas la charge de calcul — elle la quadruple.

1.1 Comment fonctionne la tokenisation par BPE

Le mécanisme standard utilisé par la plupart des LLMs s'appelle **Byte Pair Encoding** (BPE). L'algorithme a été inventé en 1994 par Philip Gage comme technique de compression de données, avant d'être adapté en 2016 par Rico Sennrich, Barry Haddow et Alexandra Birch pour la traduction automatique neuronale [13].

Le principe du BPE est élégant dans sa simplicité. On part d'un corpus d'entraînement et on considère d'abord chaque caractère comme un symbole distinct. On identifie la paire de symboles adjacents la plus fréquente, on la fusionne en un nouveau symbole, et on répète l'opération jusqu'à atteindre la taille de vocabulaire souhaitée. Les mots courants deviennent des tokens uniques (*the*, *and*), les mots rares des séquences de sous-mots (*tokenization* devient *token* + *ization* dans certains encodages GPT-4).

OpenAI utilise une implémentation open source de ce mécanisme appelée **Tiktoken** [9]. Les modèles GPT-4 et GPT-3.5 emploient l'encodage `cl100k_base` (vocabulaire de 100 000 tokens), GPT-4o l'encodage `o200k_base` (200 000 tokens). Un vocabulaire plus grand réduit le nombre de tokens nécessaires pour représenter un texte, ce qui diminue les coûts et permet de traiter des textes plus longs.

1.2 Pourquoi le français coûte plus cher que l'anglais

Le BPE a un biais structurel : il favorise les langues dominantes dans les données d'entraînement. Pour un modèle entraîné majoritairement sur de l'anglais — ce qui est le cas de tous les grands LLMs actuels, y compris Llama 3 dont 95 % du corpus est en anglais ou en code — un texte en français nécessite en moyenne 20 à 30 % de tokens supplémentaires qu'un texte anglais de même contenu sémantique. Une langue à morphologie riche (finnois, hongrois, ukrainien) peut nécessiter deux à trois fois plus de tokens.

La conséquence pratique est directe : sur les API payantes, vous payez au token. Un prompt en français consommera plus de tokens qu'un prompt équivalent en anglais. Sur des volumes importants, cet écart se traduit par des surcoûts significatifs.

1.3 Ce que le modèle voit, et ce qu'il ne voit pas

La tokenisation n'est pas neutre sur la compréhension. Des recherches publiées en 2025 dans la revue *Computational Linguistics* du MIT ont montré que des tokenisations différentes du même mot chinois produisent des représentations sémantiques différentes dans le modèle [3]. Des chercheurs ont construit un dataset adversarial (ADT, Adversarial Dataset for Tokenizer) qui démontre que des textes délibérément formulés pour perturber la tokenisation dégradent significativement les performances de GPT-4o, LLaMA-3 et DeepSeek-R1 [15].

Pour un utilisateur pratique : les fautes d'orthographe et les néologismes génèrent des tokenisations inhabituelles, potentiellement inefficaces. Certains formats numériques (timestamps, identifiants) sont particulièrement mal gérés par les tokenizers actuels, ce qui explique en partie les difficultés des LLMs à raisonner sur des données temporelles.

1.4 La fenêtre de contexte : la mémoire à court terme du modèle

Chaque LLM dispose d'une **fenêtre de contexte** (context window) exprimée en tokens. GPT-3 travaillait avec 4 096 tokens. GPT-4 Turbo a porté cette limite à 128 000 tokens. Certains modèles récents dépassent le million de tokens de contexte.

Des études ont montré que les LLMs utilisent mieux les informations placées en début et en fin de contexte qu'au milieu — un phénomène dit *lost in the middle* [16]. Mettre les instructions importantes au début du prompt, et les éléments de données juste avant la question, n'est donc pas qu'une question de style : c'est une stratégie d'efficacité.

1.5 Compter ses tokens avant d'envoyer

La bibliothèque Tiktoken permet de compter précisément le nombre de tokens d'un texte avant de l'envoyer à l'API [8]. Un texte de mille mots en français représente typiquement entre 1 200 et 1 600 tokens. Un document PDF de 50 pages peut atteindre 25 000 à 40 000 tokens. Ces ordres de grandeur permettent de dimensionner ses pipelines et d'estimer des budgets d'API.

Pour les développeurs qui n'utilisent pas OpenAI, HuggingFace propose la bibliothèque `tokenizers` qui supporte les encodages BPE, WordPiece (BERT) et Unigram (SentencePiece de Google). Les tokenizers varient selon les modèles : comparer des prix entre fournisseurs sur la seule base du coût par token sans tenir compte de la fertilité du tokenizer peut conduire à des surprises.

1.6 L'avenir sans tokenizer ?

Meta AI a publié en 2024 une architecture expérimentale appelée Byte Latent Transformer (BLT) qui opère directement sur des octets bruts, sans étape de tokenisation. L'idée : laisser le modèle apprendre lui-même à regrouper dynamiquement les octets en unités pertinentes — des « patches » — selon le contenu. Cette approche montre des résultats prometteurs sur les langues à faibles ressources et sur la robustesse aux fautes d'orthographe [11]. Elle reste expérimentale, mais illustre bien que la tokenisation est une pièce centrale et potentiellement fragile de l'architecture LLM actuelle.

2 Le chunk : découper pour retrouver

Le **chunk** est un fragment de texte. Contrairement au token — unité de traitement interne au modèle —, le chunk est une unité logique définie par l'architecte du système. On parle de chunk principalement dans le contexte du **Retrieval-Augmented Generation** (RAG), une technique qui consiste à connecter un LLM à une base de connaissances externe pour lui permettre de répondre à des questions sur des documents qu'il n'a pas vus pendant son entraînement.

Le RAG a été formalisé en 2020 par Patrick Lewis et ses collègues de Facebook AI Research dans un article fondateur publié à NeurIPS [6]. Le principe : plutôt que de charger l'intégralité d'une base documentaire dans la fenêtre de contexte — ce qui serait coûteux, voire impossible si la base dépasse la limite de contexte —, on indexe les documents sous forme de vecteurs numériques (embeddings), et on ne récupère que les fragments les plus pertinents par rapport à la question posée.

2.1 Comment fonctionne un pipeline RAG

Un pipeline RAG comprend deux phases distinctes. La phase d'indexation : les documents source sont découpés en chunks, chaque chunk est converti en vecteur numérique par un modèle d'embedding, et ces vecteurs sont stockés dans une base de données vectorielle. La phase de requête : quand un utilisateur pose une question, cette question est convertie en vecteur, et on recherche les chunks dont les vecteurs sont les plus proches sémantiquement. Ces chunks sont ensuite injectés dans le contexte du LLM qui génère sa réponse.

La qualité du résultat final dépend énormément de la qualité du découpage. Un chunk trop petit perd son contexte. Un chunk trop grand dilue le signal sémantique et charge inutilement la fenêtre de contexte.

2.2 Les stratégies de chunking et leurs compromis

Il existe plusieurs grandes familles de stratégies de découpage, chacune avec ses avantages et ses limites.

Le **chunking à taille fixe** découpe le texte en segments de N tokens avec un éventuel chevauchement (overlap) entre chunks consécutifs. C'est facile à implémenter, prévisible, et suffisant pour des documents homogènes bien structurés. Son défaut : il peut séparer une phrase de sa conclusion, ou une règle de son exception.

Le **chunking par structure logique** respecte les frontières naturelles du document (phrases, paragraphes, sections). Pour des documents bien formatés — rapports structurés, articles académiques, documentations techniques —, cette approche produit des chunks sémantiquement cohérents. Elle est moins efficace sur des textes mal formatés.

Le **chunking sémantique** utilise un modèle de langage ou des techniques de segmentation textuelle pour identifier les ruptures thématiques dans le texte, indépendamment de la structure formelle. Cette approche améliore la pertinence des chunks récupérés mais a un coût de calcul plus élevé.

Le **chunking propositionnel** décompose le texte en unités atomiques de sens : chaque chunk représente une seule affirmation, un seul fait. Cette granularité fine améliore la précision de la récupération mais multiplie le nombre de chunks.

Une étude publiée en 2025 dans PubMed Central a comparé ces quatre stratégies sur des données médicales [2]. Le chunking adaptatif a obtenu les meilleurs résultats, avec 87 % de précision médicale contre 50 % pour le chunking par taille fixe.

2.3 La taille de chunk : ordres de grandeur utiles

Il n'existe pas de taille universellement optimale. Quelques repères pratiques : pour des bases de questions-réponses courtes (FAQ, documentations de référence), des chunks de 100 à 300 tokens donnent de bons résultats. Pour des documents analytiques (rapports, études, contrats), des chunks de 500 à 1 000 tokens préservent mieux le contexte local. Pour des corpus narratifs longs, des chunks hiérarchiques — qui indexent à la fois des segments fins et des agrégats plus larges — offrent le meilleur compromis.

Le chevauchement entre chunks (overlap) est souvent sous-estimé. Mettre 10 à 20 % de tokens en commun entre deux chunks consécutifs réduit significativement le risque de perdre une information qui se trouve exactement à la frontière entre deux segments.

2.4 Les pièges classiques du chunking

Plusieurs erreurs reviennent régulièrement dans les projets RAG [5]. La première est de négliger la phase de prétraitement des documents. Un PDF mal extrait — colonnes mélangées, en-têtes inclus dans le corps du texte, tableaux transformés en suites de chiffres sans contexte — produit des chunks de mauvaise qualité quel que soit l'algorithme de découpage.

La deuxième erreur est d'oublier le contexte des chunks. Un chunk qui contient « La limite mentionnée à l'article 3.2 ne s'applique pas ici » est inutilisable sans savoir de quelle limite il s'agit. Des techniques comme le ParentDocumentRetriever de LangChain ou le SentenceWindowNodeParser de LlamaIndex adressent ce problème : on indexe de petits chunks pour la précision de la recherche, mais on injecte dans le contexte du LLM un chunk parent plus large pour préserver le contexte.

La troisième erreur est de croire qu'on peut définir une stratégie de chunking une fois pour toutes. La granularité optimale dépend du type de document, du type de requête, et du modèle d'embedding utilisé. Il faut évaluer, itérer, mesurer.

2.5 Chunking et documents financiers ou juridiques

Une étude de 2024 sur le chunking de rapports financiers SEC illustre la complexité du problème dans des contextes à forts enjeux [4]. Ces documents combinent du texte narratif, des tableaux chiffrés, des notes de bas de page et des références croisées. Un chunking naïf par taille fixe produit des chunks qui mélangent des unités hétérogènes ou isolent un chiffre de sa note explicative. Le même raisonnement s'applique aux contrats juridiques, aux cahiers des charges techniques, aux manuels de procédure.

2.6 L'approche hiérarchique : segmenter puis regrouper

Des chercheurs de l'Université de technologie de Hanoï ont publié en 2025 un framework qui combine segmentation textuelle et clustering sémantique [7]. Le document est d'abord segmenté en sections cohérentes par un modèle neuronal, puis ces segments sont regroupés en clusters basés sur leur similarité sémantique. Au moment de la requête, le système récupère à la fois des segments fins (pour la précision) et des clusters entiers (pour le contexte plus large). Évalué sur les datasets NarrativeQA, QuALITY et QASPER, ce framework surpasse les méthodes de chunking traditionnelles sur la compréhension de textes longs.

3 Ce que ça change dans vos usages quotidiens

Ces deux notions ne sont pas que des détails d'implémentation réservés aux développeurs. Elles ont des conséquences directes sur la façon dont vous interagissez avec n'importe quel outil IA.

Quand vous rédigez un prompt, vous produisez des tokens. La longueur de votre historique de conversation compte en tokens. Les fichiers que vous joignez comptent en tokens. Les exemples que vous donnez au modèle (*few-shot prompting*) comptent en tokens. Chaque token a un coût — financier sur les API payantes, computationnel sur les modèles hébergés localement.

La bonne pratique n'est pas de compresser à l'extrême. C'est de savoir ce qui est utile et ce qui ne l'est pas. Les reformulations redondantes, les politesses excessives, les rappels de contexte déjà donnés au tour précédent : tout cela consomme des tokens sans améliorer la réponse.

Si vous utilisez ou envisagez d'utiliser un système RAG, la stratégie de chunking est probablement le paramètre le plus critique de votre pipeline. Avant de choisir un modèle LLM ou un modèle d'embedding, posez-vous la question de la structure de vos documents. Sont-ils homogènes ou hétérogènes ? Courts ou longs ? Bien structurés ou en texte libre ? La réponse conditionne directement le choix de la stratégie de chunking.

Un système RAG qui répond mal n'est souvent pas un problème de LLM. C'est un problème de chunking. Améliorer le chunking améliore le système — parfois de façon spectaculaire — sans changer le modèle ni l'infrastructure.

Le token et le chunk sont deux coutures invisibles de l'infrastructure IA. Les ignorer, c'est optimiser à l'aveugle. Les comprendre, c'est gagner la capacité d'agir sur les bons paramètres quand les résultats ne sont pas à la hauteur.

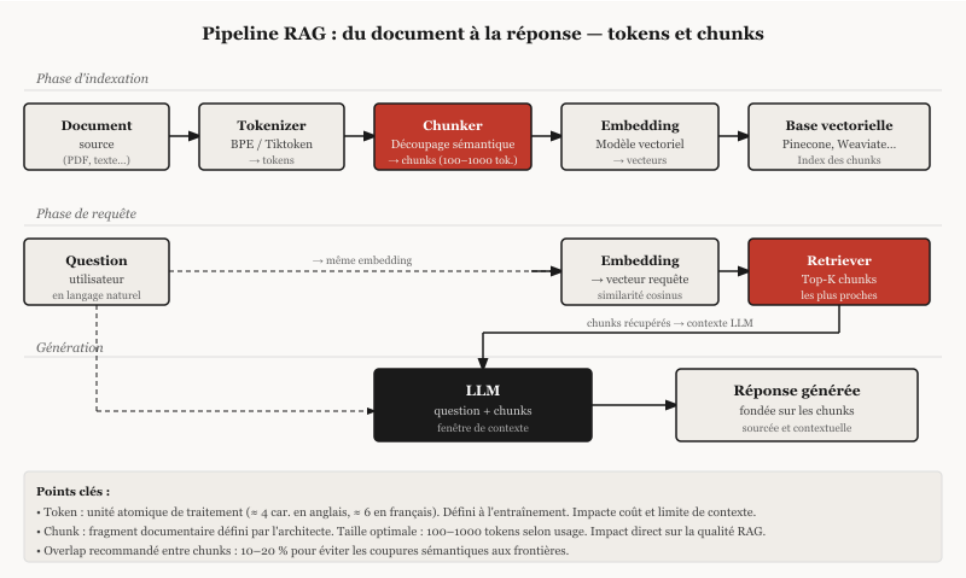

FIGURE 1 – Architecture d'un pipeline RAG : la tokenisation (tokens) conditionne les coûts et la fenêtre de contexte ; le chunking conditionne la qualité de la récupération documentaire.

Références

[1] Kaj BOSTROM et Greg DURRETT. Byte Pair Encoding is Suboptimal for Language Model Pretraining. 2020. arXiv : 2004.03720.

[2] Comparative Evaluation of Advanced Chunking for Retrieval-Augmented Generation in Large Language Models for Clinical Decision Support. In : *PubMed Central* (2025).

[3] Noah HASLETT et Yue CAI. Tokenization Changes Meaning in Large Language Models: Evidence from Chinese. In : *Computational Linguistics* 51.3 (2025), p. 785-820.

[4] Antonio JIMENO YEPES et al. Financial Report Chunking for Effective Retrieval Augmented Generation. 2024. arXiv : 2402.05131.

[5] Aadit KSHIRSAGAR. Enhancing RAG Performance Through Chunking and Text Splitting Techniques. In : *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 10.5 (2024), p. 151-158.

[6] Patrick LEWIS et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In : *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. 2020.

[7] Hoang Thanh NGUYEN, Tri Duc NGUYEN et Van Huy NGUYEN. Enhancing Retrieval Augmented Generation with Hierarchical Text Segmentation Chunking. In : *Information and Communication Technology. SOICT 2024*. T. 2352. Communications in Computer and Information Science. Singapore : Springer, 2025.

[8] OPENAI. How to count tokens with Tiktoken. 2024.

[9] OPENAI. Tiktoken – Fast BPE tokeniser for use with OpenAI's models. 2024.

[10] OPENAI. What are tokens and how to count them? 2024.

[11] Artidoro PAGNONI et al. Byte Latent Transformer: Patches Scale Better Than Tokens. Meta AI Research. 2024.

[12] Nived RAJARAMAN, Jiantao JIAO et Kannan RAMCHANDRAN. Toward a Theory of Tokenization in LLMs. 2024. arXiv : 2404.08335.

[13] Rico SENNRICH, Barry HADDOW et Alexandra BIRCH. Neural Machine Translation of Rare Words with Subword Units. In : *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*. Berlin, Germany : Association for Computational Linguistics, 2016, p. 1715-1725.

[14] Ashish VASWANI et al. Attention Is All You Need. In : *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017.

[15] Dixuan WANG et al. Tokenization Matters! Degrading Large Language Models through Challenging Their Tokenization. 2024. arXiv : 2405.17067.

[16] Xindi Wang et al. Beyond the Limits: A Survey of Techniques to Extend the Context Length in Large Language Models. 2024. arXiv : 2402.02244.