

L'inefficience des tokenizers BPE sur les langages symboliques : une étude empirique et une solution simple

Adrien Cros¹

Ava^{1,*}

¹Avalon Research, Indépendant

*Agent IA — Claude Opus 4.6

contact@avalon-network.com

Février 2026

Résumé

Les tokenizers Byte-Pair Encoding (BPE), conçus pour le langage naturel, fragmentent systématiquement les symboles composés des langages symboliques spécialisés en 2 à 5 sous-tokens, annulant les gains de compression que ces notations sont censées apporter. Nous mesurons ce phénomène sur un corpus de 43 paires (langage naturel français ↔ notation symbolique .ava) en utilisant le tokenizer Qwen 2.5 (151 665 tokens). Nous montrons qu'ajouter seulement 26 tokens domain-specific au vocabulaire — soit une augmentation de 0,017 % — améliore la compression moyenne de **112,4 %** (de $2,65\times$ à $5,62\times$). Le gain est concentré sur les catégories dont la notation repose sur des composés multi-symboles (jusqu'à +318,9 % pour les cicatrices cognitives). Appliqué à une fenêtre de contexte de 200K tokens, cela représente un gain de 595K tokens effectifs. Nous analysons pourquoi les symboles Unicode individuels sont déjà bien gérés par les vocabulaires modernes, mais pas leurs combinaisons, et proposons l'extension ciblée du vocabulaire comme solution triviale à ce problème sous-estimé.

Mots-clés : tokenization, BPE, langages symboliques, compression, extension de vocabulaire, notation formelle, LLM

1 Introduction

L'algorithme Byte-Pair Encoding (BPE), introduit pour la traduction automatique par Sennrich et al. [1], est devenu le standard de facto pour la tokenization dans les grands modèles de langage

(LLM). Son principe — fusionner itérativement les paires de symboles les plus fréquentes dans un corpus d'entraînement — produit des vocabulaires efficaces pour le langage naturel, où les distributions de sous-mots sont relativement stables entre les langues.

Cependant, les vocabulaires BPE sont construits à partir de corpus dominés par le langage naturel. Lorsqu'un LLM doit traiter un langage symbolique spécialisé — notation chimique (SMILES), musicale (ABC), mathématique, ou toute notation de domaine — les symboles composés de cette notation n'apparaissent pas dans le corpus d'entraînement du tokenizer. Ils sont donc fragmentés en sous-tokens, parfois de manière pathologique.

Ce problème est rarement quantifié. Dans ce travail, nous l'étudions empiriquement sur un cas concret : la notation **.ava**, un langage symbolique compact utilisant des symboles Unicode pour représenter des structures psychologiques complexes. Nous mesurons la tokenization avant et après l'ajout de 26 tokens domain-specific, et montrons que cette intervention triviale — 0,017 % du vocabulaire — produit un gain de compression de 112,4 %.

2 Contexte

2.1 Tokenization par sous-mots

Trois algorithmes dominent la tokenization des LLM modernes :

- **BPE** [1] : fusion itérative des paires les plus fréquentes. Utilisé par GPT, LLaMA, Qwen,

Mistral.

- **Unigram LM** [2] : modèle probabiliste qui sélectionne la segmentation la plus probable. Utilisé par T5, mBART.
- **SentencePiece** [2] : framework unifiant BPE et Unigram, opérant directement sur le texte brut sans pré-tokenization.

Les vocabulaires modernes varient de 32K (LLaMA) à 151K (Qwen 2.5) tokens. Leur construction repose sur des corpus massifs de texte naturel, de code, et de données web. Les symboles Unicode rares mais standards (caractères mathématiques, flèches, symboles techniques) sont généralement inclus comme tokens individuels.

2.2 Alternatives au tokenizer

Des travaux récents explorent des approches sans tokenizer : MegaByte [3] opère directement au niveau des bytes, éliminant le besoin d'un vocabulaire fixe. Ces approches restent marginales en production, et la grande majorité des LLM déployés utilisent BPE.

2.3 La notation .ava

La notation **.ava** est un langage symbolique conçu pour représenter de manière compacte des profils psychologiques : traits de personnalité, biais cognitifs, cicatrices émotionnelles, contraintes, projets et relations. Elle utilise des symboles Unicode comme préfixes de catégorie :

- @ — personnes et rôles
- Ψ — traits psychologiques
- ∅ — cicatrices (patterns évités)
- ∠ — biais cognitifs
- ! — contraintes et règles
- Δ — projets
- ◇ — infrastructure

Chaque entrée combine un préfixe symbolique avec un identifiant textuel compact, parfois suivi de modificateurs : **∅dns9!** (cicatrice d'attaque DNS, priorité haute), **∠over-eng** (biais de sur-ingénierie).

3 Méthodologie

3.1 Corpus

Notre corpus se compose de 43 paires, chacune contenant :

1. Une description en langage naturel français.
2. Son équivalent en notation **.ava**.

Les paires couvrent six catégories sémantiques : personnes (@), cicatrices (∅), biais (∠), contraintes (!), projets (Δ) et infrastructure (◇).

3.2 Tokenizer de base

Nous utilisons le tokenizer du modèle Qwen 2.5-1.5B-Instruct, implémenté via la bibliothèque HuggingFace **transformers**. Ce tokenizer possède un vocabulaire de **151 665 tokens**, parmi les plus larges des modèles open-source actuels.

3.3 Extension du vocabulaire

Nous ajoutons 26 tokens au vocabulaire via la méthode **add_tokens()** de HuggingFace :

- **18 symboles .ava** — les symboles Unicode et identifiants de base de la notation.
- **8 composés fréquents** — les combinaisons symbole+identifiant les plus courantes dans le corpus (ex. : **∅dns9!**, **∠over-eng**, **∅ghost**).

Le vocabulaire étendu compte **151 690 tokens**, soit une augmentation de 0,017 %.

3.4 Mesure de compression

Pour chaque paire, nous calculons le ratio de compression :

$$R = \frac{N_{NL}}{N_{.ava}} \quad (1)$$

où N_{NL} est le nombre de tokens de la description en langage naturel et $N_{.ava}$ le nombre de tokens de la notation **.ava**. Nous mesurons R avec le tokenizer de base et le tokenizer étendu.

4 Résultats

4.1 Compression globale

Le tableau 1 présente les statistiques de compression sur l'ensemble du corpus.

Tab. 1 : Compression globale (43 paires)

Métrique	Base	Étendu
Moyenne	2,65×	5,62×
Médiane	1,93×	2,08×
Maximum	9,0×	36,0×
Écart-type	1,68	8,34
Δ moyenne	+112,4 %	
Δ médiane	+7,7 %	

4.2 Compression par catégorie

Le tableau 2 détaille les résultats par catégorie sémantique.

Tab. 2 : Compression par catégorie

Catégorie	Sym.	Base	Étendu	Δ
Cicatrices	∅	6,62×	27,75×	+318,9%
Biais	∠	4,36×	15,67×	+259,2%
Contraintes	!	3,56×	5,56×	+56,1%
Personnes	@	2,46×	2,70×	+9,4%
Projets	△	1,89×	1,94×	+2,8%
Infrastructure	◇	1,74×	1,74×	+0,0%

4.3 Divergence moyenne-médiane

Un résultat frappant est la divergence entre l'amélioration de la moyenne (+112,4%) et celle de la médiane (+7,7%). Cette divergence s'explique par la distribution bimodale des gains :

- Les catégories à **faible densité symbolique** (personnes, projets, infrastructure) contiennent principalement des identifiants textuels standards que le tokenizer de base gère déjà correctement. Leur compression change peu.
- Les catégories à **haute densité symbolique** (cicatrices, biais) contiennent des composés multi-symboles que le tokenizer de base fragmente en 4-5 sous-tokens. L'extension les réduit à 1 token, produisant des gains spectaculaires.

La médiane, dominée par les catégories à faible densité symbolique (plus nombreuses dans le corpus), reflète le cas typique. La moyenne capture l'impact disproportionné des catégories symboliquement denses.

4.4 Impact sur la fenêtre de contexte

Avec une fenêtre de contexte de 200 000 tokens (standard pour les modèles modernes), la compression moyenne passe de :

- **Base** : $200K \times 2,65 = 529K$ tokens effectifs
- **Étendu** : $200K \times 5,62 = 1\,124K$ tokens effectifs

Soit un gain de **595 000 tokens effectifs**, ou l'équivalent de ~ 450 pages de texte supplémentaires dans la même fenêtre de contexte.

5 Analyse

5.1 Le paradoxe des symboles individuels

Un résultat contre-intuitif émerge de l'analyse au niveau des symboles individuels : **la plupart des symboles Unicode utilisés par .ava sont déjà des tokens uniques dans le vocabulaire**

Owen 2.5. Les caractères Ψ , \emptyset , Δ , \diamond , \angle sont tous tokenisés en un seul token par le tokenizer de base.

Seuls trois symboles individuels bénéficient de l'extension :

- Ψ : 2 tokens \rightarrow 1 token
- T^2 : 2 tokens \rightarrow 1 token
- $\$!i$: 3 tokens \rightarrow 1 token

Ce résultat montre que les vocabulaires BPE modernes, avec 100K+ tokens, couvrent bien l'espace Unicode standard. **Le problème n'est pas les symboles, mais leurs combinaisons.**

5.2 Le vrai goulot : les composés

Le gain de compression provient quasi-exclusivement des **tokens composés** : des séquences comme $\emptyset dns9!$, $\angle over-eng$, $\emptyset ghost$ qui combinent un symbole Unicode, un identifiant textuel, et parfois un modificateur.

Le tokenizer BPE de base fragmente ces composés parce qu'il ne les a jamais vus dans son corpus d'entraînement. L'algorithme BPE ne peut fusionner que des paires de tokens qu'il a observées ; une séquence comme $\emptyset dns9!$ est traitée comme 4-5 tokens indépendants : \emptyset , d , ns , 9 , $!$.

En ajoutant ces composés comme tokens atomiques, chaque occurrence passe de 4-5 tokens à exactement 1, ce qui explique les gains de +318,9% sur les cicatrices (catégorie utilisant les

composés les plus longs et les plus fréquents).

5.3 Pourquoi l'infrastructure ne gagne rien

La catégorie infrastructure (\diamond) montre un gain de 0,0 %, ce qui constitue un contrôle négatif naturel. Les entrées d'infrastructure utilisent principalement des noms de serveurs et des chemins de fichiers — du texte technique standard que le tokenizer BPE gère déjà efficacement. Le symbole \diamond lui-même est déjà un token unique, et les identifiants qui le suivent sont des mots courants dans les corpus de code.

6 Implications

6.1 Ratio coût/bénéfice

L'ajout de 26 tokens à un vocabulaire de 151 665 représente une augmentation de 0,017 %. En termes pratiques :

- La matrice d'embedding du modèle augmente de 26 lignes (négligeable en mémoire).
- Aucun ré-entraînement du modèle n'est nécessaire ; les nouveaux tokens sont initialisés aléatoirement et affinés pendant le fine-tuning.
- Le gain de compression est de +112,4 % en moyenne.

Le ratio bénéfice/coût est de l'ordre de $112/0,017 \approx 6\,600\times$: chaque pourcent d'augmentation du vocabulaire produit $\sim 6\,600\%$ de gain de compression. Ce ratio exceptionnel suggère que les vocabulaires BPE standards sont sévèrement sous-optimaux pour les notations symboliques.

6.2 Généralisation à d'autres domaines

Le même phénomène devrait se manifester pour tout langage symbolique dont les composés n'apparaissent pas dans les corpus d'entraînement BPE :

- **SMILES** (chimie) : les chaînes moléculaires comme `CC(=O)Oc1ccccc1C(=O)O` combinent des caractères courants en séquences spécialisées.
- **Notation musicale** (ABC, LilyPond) : les séquences de notes et d'accords forment des composés absents des corpus textuels.

- **Expressions régulières** : les motifs complexes comme `(?<=d){3}` sont fragmentés en sous-tokens individuels.
- **Formules mathématiques** : les expressions LaTeX composées (`\frac{d}{dx}`) pourraient bénéficier d'une tokenization atomique.

6.3 Recommandation pratique

Pour tout pipeline de fine-tuning impliquant un langage symbolique :

1. Identifier les N composés les plus fréquents du domaine.
2. Les ajouter au vocabulaire via `add_tokens()`.
3. Redimensionner la matrice d'embedding (`resize_token_embeddings()`).
4. Fine-tuner normalement.

Le coût est nul en complexité d'implémentation, négligeable en calcul, et les gains potentiels sont substantiels.

7 Travaux connexes

BPE pour la NMT. Sennrich et al. [1] ont introduit BPE pour la traduction automatique, résolvant le problème du vocabulaire ouvert en segmentant les mots rares en sous-unités. Cette approche a été universellement adoptée par les LLM.

SentencePiece et Unigram. Kudo [2] a proposé SentencePiece, un framework de tokenization indépendant de la langue, incluant l'algorithme Unigram LM comme alternative probabiliste à BPE. Ces approches partagent la même limitation fondamentale : le vocabulaire est construit à partir du corpus d'entraînement.

Modèles byte-level. Yu et al. [3] ont proposé MegaByte, un modèle opérant directement sur des bytes, éliminant le besoin d'un tokenizer. Cette approche résout théoriquement le problème de fragmentation mais au prix d'une complexité computationnelle accrue.

Adaptation de vocabulaire. Plusieurs travaux ont exploré l'extension de vocabulaire pour l'adaptation à de nouvelles langues [4, 5]. Notre approche diffère en ciblant non pas une langue naturelle mais un langage symbolique, et en montrant qu'un nombre extrêmement réduit de tokens suffit.

Tokenization et performance des modèles.

Rust et al. [6] ont montré que la qualité de la tokenization affecte directement la performance des modèles multilingues, les langues sous-représentées dans le vocabulaire souffrant d’une fragmentation excessive. Notre travail étend cette observation aux langages symboliques artificiels.

8 Limites

Plusieurs limites importantes doivent être soulignées :

1. **Taille du corpus.** Avec 43 paires, notre corpus est petit. Les résultats statistiques (notamment l’écart-type de 8,34 après extension) reflètent cette limitation.
2. **Un seul domaine.** Nous n’étudions que la notation `.ava`. La généralisation à d’autres langages symboliques reste à démontrer empiriquement.
3. **Compression seule.** Nous mesurons uniquement la compression (nombre de tokens), pas l’impact sur la qualité des sorties du modèle. Un tokenizer plus compressif n’est bénéfique que si le modèle apprend correctement les nouveaux tokens pendant le fine-tuning.
4. **Concentration des gains.** Le gain de +112,4% en moyenne masque une distribution très inégale : les catégories cicatrices et biais tirent la moyenne vers le haut, tandis que trois catégories sur six voient des gains inférieurs à 10%.
5. **Pas de comparaison inter-tokenizers.** Nous n’avons testé que le tokenizer Qwen 2.5. D’autres tokenizers (LLaMA, GPT-4) pourraient se comporter différemment, notamment ceux avec des vocabulaires plus petits.

9 Conclusion

Le tokenizer est un composant souvent traité comme une boîte noire dans les pipelines de LLM. Nos résultats montrent qu’il peut constituer un goulot d’étranglement significatif pour les langages symboliques : le tokenizer BPE de Qwen 2.5, malgré son vocabulaire de 151 665 tokens, fragmente les composés symboliques de la notation `.ava` en 4–5 sous-tokens.

L’ajout de 26 tokens — 0,017% du vocabulaire — élimine cette fragmentation et produit un gain de compression moyen de 112,4%. Ce résultat met en lumière un problème structurel de BPE : l’algo-

ritme ne peut pas apprendre des composés qu’il n’a jamais vus, même si chaque symbole constituant est déjà dans le vocabulaire. La solution — ajouter les composés fréquents du domaine — est triviale à implémenter et ne devrait être omise dans aucun pipeline de fine-tuning impliquant une notation spécialisée.

Plus largement, ce travail suggère que l’adaptation du tokenizer est un levier sous-exploité dans le fine-tuning de LLM, offrant des gains disproportionnés par rapport à son coût.

Références

- [1] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proc. ACL*, 2016, pp. 1715–1725.
- [2] T. Kudo and J. Richardson, “SentencePiece : A simple and language independent subword tokenizer and detokenizer for neural text processing,” in *Proc. EMNLP*, 2018, pp. 66–71.
- [3] L. Yu, D. Simig, C. Flaherty, A. Aghajanyan, L. Zettlemoyer, and M. Lewis, “MEGABYTE : Predicting million-byte sequences with multiscale transformers,” in *Proc. NeurIPS*, 2023.
- [4] Z. Wang, K. Mayhew, D. Roth, et al., “Extending multilingual BERT to low-resource languages,” in *Findings of EMNLP*, 2020.
- [5] E. Chau, L. Lin, and N. A. Smith, “Parsing with multilingual BERT, a small corpus, and a small treebank,” in *Findings of EMNLP*, 2020.
- [6] P. Rust, J. Pfeiffer, I. Vulić, S. Ruder, and I. Gurevych, “How good is your tokenizer? On the monolingual performance of multilingual language models,” in *Proc. ACL*, 2021, pp. 3118–3135.