A Triadic Suffix Tokenization Scheme for Numerical Reasoning

Olga Chetverina*

April 20, 2026

Abstract

Standard subword tokenization methods fragment numbers inconsistently, causing large language models (LLMs) to lose positional and decimal structure - a primary driver of errors in arithmetic and scientific reasoning.

We introduce Triadic Suffix Tokenization (TST), a deterministic scheme that partitions digits into three-digit triads and annotates each triad with an explicit magnitude marker. Critically, the scheme defines a fixed, one-to-one mapping between suffixes and orders of magnitude for the integer part (thousands, millions, billions, etc.) and a parallel system of replicated markers for fractional depth (tenths, thousandths, millionths, etc.). Unlike approaches that rely on positional inference, this method provides a consistent gradient signal, which should ensure stable convergence.

Two implementation variants are proposed: (1) a vocabulary-based approach that adds at most 10,000 fixed tokens to an existing vocabulary, covering 33 orders of magnitude (10^{-15} to 10^{18}); and (2) a suffix-marker approach that uses a small set of special tokens to denote magnitude dynamically. Both variants preserve exact digits while making order-of-magnitude relationships transparent at the token level. While we focus on 3-digit groups (Triadic), the framework is inherently scalable to any group size for precise vocabulary optimization. Furthermore, it allows for linear vocabulary expansion to accommodate arbitrary precision and range. TST is architecture-agnostic and can be integrated as a drop-in preprocessing step. Experimental validation is deferred to future work.

1 Introduction

Large Language Models (LLMs) have achieved remarkable performance on complex reasoning tasks, yet they frequently stumble on basic numerical understanding. The famous "$9.11 > 9.9$" failure exemplifies how models misjudge decimal magnitudes [3]. This weakness stems largely from tokenization: numbers are fragmented into arbitrary subword units, losing their positional and magnitude information [4, 10]. Standard tokenizers split "100400" into "100" and "400" without encoding that the former represents hundreds of thousands. As a result, models must learn magnitude relationships from scratch — a statistically inefficient task.

Recent work has explored various solutions: specialized number tokens [6], continuous number encodings such as xVal [11], right-to-left tokenization with comma separation [5], and comprehensive benchmarks like NumericBench [1] and Number Cookbook [3]. Studies comparing base-10 (digit-level) versus base-1000 (triad-level) tokenization show that digit-level approaches are more data-efficient, but struggle with magnitude comprehension [4, 5].

This paper introduces a simple yet effective tokenization scheme that combines triad grouping (base-1000) with explicit magnitude annotations for both integer and fractional parts, providing a stronger inductive bias for numerical reasoning.

*Lecturer at MIPT, Moscow, Russia. Email: tst.chetverina@gmail.com. ORCID: 0000-0003-4924-8130. Previously archived at Zenodo: DOI 10.5281/zenodo.18999577. This research was conducted independently outside of any institutional assignments, and no institutional resources were used.

2 Related Work

Digit-level tokenization (base-10) treats each digit as an independent token. While data-efficient, it lacks explicit magnitude cues, forcing models to infer order from positional patterns alone [5]. Multi-digit tokenization, used by GPT-3.5 and GPT-4, reduces sequence length but introduces arbitrary boundaries that can fragment numbers inconsistently [4].

xVal encodes numbers as continuous embeddings using a single token per number, demonstrating strong interpolation performance [11]. However, it discards exact digits, trading precision for smoothness—a trade-off unsuitable for tasks requiring exact arithmetic.

Right-to-left tokenization, which adds commas every three digits from the right, has been shown to improve integer arithmetic by up to 20% compared to unformatted numbers [4,5]. This demonstrates that even simple formatting shifts help models reason about numbers. However, commas only group digits; they do not indicate the magnitude of each group. A model still must infer whether "123" stands for 123, 123,000, or 123,000,000 from its position in the sequence.

Recent work has also explored modifying the loss function rather than tokenization. Number Token Loss (NTL) [9] replaces the standard cross-entropy loss with a regression-like objective for numerical tokens, treating numbers as continuous values during training. This approach preserves exact digits while providing a smooth gradient signal. Importantly, NTL operates at the training level and is orthogonal to tokenization schemes—it can be combined with any input representation.

Several established benchmarks can assess numerical reasoning in future studies: NumericBench evaluates six fundamental numerical capabilities [1], Number Cookbook covers 17 distinct numerical tasks [3], and studies probing enumeration skills show that even state-of-the-art models lack systematic enumeration [2].

3 Proposed Method: Triadic Suffix Tokenization (TST)

3.1 Core Principles

1. Group digits into triads (thousands, millions, etc.).
2. Annotate each triad with explicit magnitude markers.
3. Preserve exact digits.

3.2 Integer Part

Digits are grouped from right to left:

$$\begin{aligned} 100400 &\rightarrow 100\text{k}\,400 \\ 1234567 &\rightarrow 1\text{m}\,234\text{k}\,567 \\ 100200400 &\rightarrow 100\text{m}\,200\text{k}\,400 \\ 123456789012345 &\rightarrow 123\text{t}\,456\text{b}\,789\text{m}\,012\text{k}\,345 \end{aligned}$$

3.2.1 Why Suffixes, Not Just Commas

Unlike right-to-left commas, which only group digits, the suffixes k, m, b, t, q explicitly indicate the order of magnitude. This gives the model direct access to the scale of each digit group, rather than forcing it to infer magnitude from position.

Suffix	Magnitude	Power
k	thousand	10^3
m	million	10^6
b	billion	10^9
t	trillion	10^{12}
q	quadrillion	10^{15}

Table 1: Integer suffixes

3.3 Fractional Part

Fractional digits are grouped left to right with replicated 'p' markers. To ensure a canonical representation and maintain a 1:1 mapping between tokens and numerical values, all fractional triads are **right-padded with trailing zeros** to a fixed length of three digits. This normalization ensures that numerically equivalent representations (e.g., 0.1 and 0.100) result in the same token sequence.

$$\begin{aligned} 1.12345678 &\rightarrow 1.\ 123p\ 456pp\ 780ppp \\ 0.0045 &\rightarrow 0.\ 004p\ 500pp \\ \pi &\rightarrow 3.\ 141p\ 592pp\ 653ppp\ 589pppp\ 793ppppp \end{aligned}$$

Maximum practical depth is typically 5 'p's (15 decimal places). Without padding, the vocabulary would unnecessarily expand to include sub-triadic variations, and the model would be forced to learn that $0.1, 0.10$, and 0.100 represent the same value independently. Padding ensures that each token consistently represents a value in the form $n \times 10^{-3k}$ where $n \in [0, 999]$, requiring only 1,000 tokens for each triad depth.

This normalization prioritizes numerical consistency and computational precision over the preservation of the original surface form. While mapping numerically equivalent values to a single sequence improves convergence, some tasks require trailing zeros to carry semantic meaning. A detailed discussion of this trade-off and a proposed minor extension for precision-preserving applications are provided in Section 6.2.

3.4 Vocabulary Considerations

Two options are possible:

- **Option A (Separate tokens):** Keep digit groups and suffixes as separate tokens. Vocabulary adds only 10 new tokens (k, m, b, t, q, p, pp, ppp, pppp, ppppp). This minimally expands the vocabulary; models may learn to combine digits with suffixes.
- **Option B (Compound tokens):** Create combined tokens like "100k", "234m". With triads from 000–999 and 5 integer suffixes: $1000 \times 5 = 5000$ tokens. For fractions with up to 5 'p' depths and triads 000–999: another 5000 tokens. Total 10,000 new tokens — manageable for modern vocabularies.

Option B produces shorter input sequences (lower *token count*) and presents the model with ready-made magnitude-digit units, eliminating any ambiguity about which suffix belongs to which digit group. The smaller vocabulary of Option A is appealing, but the trade-off between sequence length and vocabulary size remains an empirical question.

While we focus on $N = 3$ (Triadic) for its alignment with human readability and its effective balance between vocabulary size and sequence length, the framework is fundamentally designed for any group size N, allowing for precise optimization of the model's vocabulary footprint. It

is worth noting that Option B produces the exact same number of input tokens as a simple division into groups of three. Even in the extreme case of $N = 1$, the input context length remains no larger than that of standard digit-by-digit representations, as each digit is fused with its magnitude marker into a single atomic token. While Option A is more "token-greedy" because it emits separate markers, it may still be practical for comparatively larger group sizes ($N \geq 3$) where the relative overhead becomes negligible.

The complete deterministic procedure for the TST scheme, illustrating the generation of magnitude-aware compound tokens as described in Option B, is formalized in Algorithm 1. For Option A, the algorithm remains identical except in Step 4, where the digit group and the suffix are appended as two separate tokens instead of a single compound unit.

Algorithm 1 Generalized Suffix Tokenization (GST) Mapping

Require: A raw numerical string S, and internal group size N optimized for the model's vocabulary.

Ensure: A sequence of magnitude-aware tokens T

// Step 1: Normalization and Cleansing
Prefix $\leftarrow$ ExtractPrefix(S)
$S' \leftarrow$ NormalizeToStandardDigits(S)
$I, F \leftarrow$ SplitIntoIntegerAndFractionalParts(S')
// Step 2: Integer Processing
$\mathcal{I} \leftarrow$ DivideIntoGroupsOfSizeFromRightToLeft(I, N)
$I' \leftarrow$ PadWithZerosFromLeftToMultipleOf($\mathcal{I}, N$)
// Step 3: Canonical Fraction Processing
$\mathcal{F} \leftarrow$ DivideIntoGroupsOfSizeFromLeftToRight(F, N)
$F' \leftarrow$ PadWithZerosFromRightToMultipleOf($\mathcal{F}, N$)
// Step 4: Tokens Generation with Suffixes
$T \leftarrow [\,]$
if Prefix is not empty **then**
    T.append(Prefix)
end if
// Process Integer Parts (Option B depicted)
for each group G_k **in** $\mathcal{I}$ at magnitude order k **do**
    suffix $\leftarrow$ GetMagnitudeSuffix(k)
    T.append(G_k + suffix)
end for
T.append(".")
// Process Fractional Parts
for each group G_p **in** $\mathcal{F}$ at decimal depth p **do**
    suffix $\leftarrow$ GetPrecisionSuffix(p)
    T.append(G_p + suffix)
end for
return T

4 Annotation Placement: Prefix vs. Suffix

A related line of work, NumeroLogic [8], adds a prefix indicating the total number of digits in a number (e.g., `<sn>2<mn>42<en>` for 42). On short-number arithmetic (addition of numbers up to three digits), this approach improves accuracy, demonstrating that explicit structural annotations help. However, NumeroLogic provides a single prefix per number and does not annotate individual triads. Its effectiveness on longer numbers or fractional precision tasks remains an open question.

TST, in contrast, is designed to scale across 33 orders of magnitude (10^{-15} to 10^{18}) by annotating each triad with its magnitude. Given the success of prefix-based length annotation on short numbers, one might expect similar or stronger benefits from TST's more detailed per-triad annotation across the full numeric range.

Beyond the choice between a single length prefix and per-triad suffixes, there is also the question of where to place the marker within each triad: before the digits (prefix) or after (suffix). For example, the number 123,456 could be represented as:

- **Suffix format**: `123k 456`
- **Prefix format**: `k123 456`

When using compound tokens (Option B: `123k` or `k123` as a single token), the distinction between prefix and suffix disappears at the token level—both become atomic vocabulary units, and the embedding layer treats them identically regardless of internal order.

However, when numbers are tokenized digit by digit, suffix placement offers several advantages:

- **Deterministic boundaries**: The suffix marks the end of a triad. For `1 2 3k 4`, the model knows that "1 2 3" form a complete thousand group before seeing further digits. With a prefix (`k 1 2 3 4`), the model cannot know where the group ends.
- **Alignment with human reading**: Suffix notation (e.g., "123k") matches the natural way numbers are written with separators (123,000) and spoken ("one hundred twenty-three thousand").
- **Implicit length information**: A suffix like `q` (quadrillion) tells the model that exactly 15 more digits (five triads) follow, providing a strong prior for the total length of the integer part.

Prefix placement could potentially provide the magnitude information slightly earlier, but it does so at the cost of slight boundary ambiguity. For these reasons, we adopt the **suffix** placement for TST when digit-level tokenization is used. In the compound-token variant (Option B), the choice is immaterial. The suffix versus prefix question remains an empirical one for some cases, but the deterministic boundary property makes suffix a natural and robust default.

5 Statistical Analysis

5.1 Comparison Framework

The comparison of different approaches is provided in the Table 2.

5.2 Flexibility: Suffixes with Digit-Level Tokenization

Even if one chooses to tokenize numbers as individual digits (base-10), suffixes can still provide magnitude cues without sacrificing digit-level precision. Consider the number 1,234,567:

- **Pure digit-level**: `1 2 3 4 5 6 7` — no magnitude information.
- **Digit-level with suffixes**: `1m 2 3 4k 5 6 7` — explicit magnitude hints for each triad, preserving exact digits while adding structural cues.

This hybrid approach increases sequence length compared to triad-level grouping but offers a flexible trade-off: the model receives exact digits (preserving precision) alongside explicit magnitude annotations (improving reasoning). Suffixes thus enable designers to choose between compact sequences (triad-level) or maximal digit-level precision with structural hints—a flexibility not offered by pure digit-level or pure group-level schemes.

Table 2: Comparison of tokenization schemes: inductive biases.

Scheme	Exact digits	Magnitude info	Sequence length	Inductive bias
Digit-level (base-10)	✓ preserved	Implicit (must learn)	Long	Digits independent
xVal [11]	× lost	Continuous encoding	1 token per number	Smoothness
Right-to-left [4]	✓ preserved	Positional only	Medium	Place-value alignment
TST (proposed)	✓ preserved	✓ Explicit	Medium	Explicit hierarchy

5.3 Hypothesis

Explicit magnitude annotation provides a stronger inductive bias than implicit or continuous approaches. TST preserves exact digits while making order explicit, potentially offering the best of both worlds. The suffix-based design also enables graceful trade-offs between sequence length and precision, depending on downstream requirements.

6 Discussion

6.1 Advantages

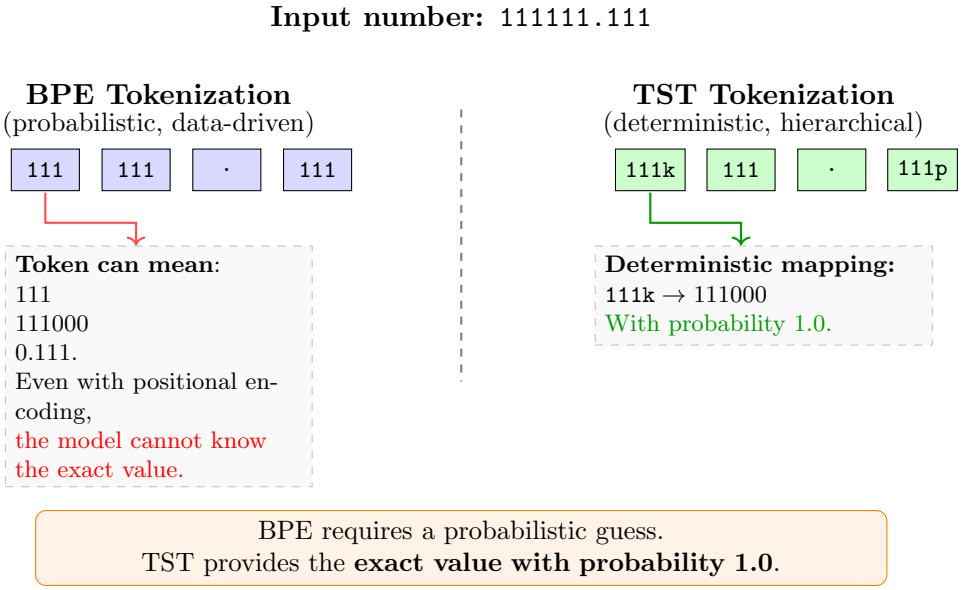

Figure 1: Comparison: BPE (left) vs TST Option B (right) for tokenizing 111111.111.

The following advantages primarily reflect Option B representation, though many also apply to Option A features.

- **Explicit real token value**: As shown in Figure 1 unlike other approaches, TST in option B establishes a bijective mapping between the token and its real numerical value, ensuring no ambiguity in decoding.

- **Fractional precision**: It guarantees that the model understands the decimal part since replicated 'p' markers create an ordinal scale of decimal depth.

- **Compatibility**: TST requires only a different tokenizer; no modifications to the model architecture are needed. While new token embeddings must be learned, the core model remains unchanged. Moreover, for Option B (compound tokens), the choice of suffix symbol is immaterial — each combined token receives its own embedding, eliminating any risk of linguistic conflict (e.g., between suffix "k" and the letter "k" in text). Option A may require reserved tokens as $[MAG_k], [MAG_p]$ etc. to avoid such conflicts.

- **Complementarity with NTL**: TST operates at the input level (tokenization), while NTL [9] modifies the loss function to incorporate numerical closeness. The two are orthogonal and can be combined: TST provides explicit structural information at the input, and NTL adds a numerical inductive bias during training. Their synergy may yield stronger improvements than either approach alone.

- **Stable convergence**: We hypothesize that explicit real token values provide a consistent gradient signal, leading to faster and more stable convergence. By removing the need for the model to infer numerical magnitude from context, TST would likely reduce both inference errors and the computational cost of training.

- **Scalability to arbitrary precision**: The TST method's current 33-order magnitude range is fully scalable, allowing for extension to arbitrary ranges by defining additional suffix tokens without changing the core triadic logic. This flexible token budget allows the framework to adapt to specific domain requirements, spanning from quantum physics to astronomy, simply by adjusting the vocabulary size. Extending TST to cover an additional three orders of magnitude requires exactly 1000 new tokens — all triads 000–999 with the new suffix.

- **Canonical fractions**: Zero padding ensures that 0.1, 0.10, and 0.100 all map to the same token sequence $0.\ 100p$, unlike standard tokenization where they differ. This eliminates surface-form ambiguity for numerically equivalent numbers. For example, even for Option A, comparison of $0.19 \to 190p$ and $0.9 \to 900p$ becomes obvious.

6.2 Limitations and Future Work

- **Experimental validation**: This paper establishes the theoretical framework and algorithmic foundation for TST. Although in a production environment (Option B) numbers are mapped directly to magnitude-aware tokens, the suffix-based notation is effective during the experimental stage. It allows for a modular testing workflow: a researcher can implement the parser separately and simply expand the model's vocabulary, evaluating the scheme's impact without deep modifications to the underlying tokenization engine. A reference implementation of the TST scheme for both Option A and Option B and the resulting vocabulary for Option B are provided for reproducibility at [12]. It is intended to demonstrate the triadic suffix mapping and is not yet an exhaustive library for all numerical notation. Future work should evaluate on numerical reasoning benchmarks like NumericBench [1] and arithmetic tasks, comparing against digit-level, xVal, right-to-left [4, 5], and NumeroLogic [8].

- **Tokenization determinism**: Numbers written without separators are processed unambiguously: integer parts are grouped from the right, fractional parts from the left. Numbers with fewer than four digits receive no suffix. This deterministic procedure works for any numeric string and requires no complex heuristics.

- **Padding fractional parts:** Normalizing fractional parts to three digits discards trailing zeros, mapping numerically equivalent values (e.g., $0.1, 0.10, 0.100$) to a single token sequence (`100p`). For tasks where the original surface form carries semantic meaning — such as fixed-point financial data or significant figures in scientific records — this loss of representational information may be undesirable.

  To address this, the canonical 1,000-token scheme can be augmented with length-specific terminator tokens: $0.1 \rightarrow$ `0. 100p [T1]`, and $0.10 \rightarrow$ `0. 100p [T2]`. In this case, the numerical part remains identical, eliminating the need to expand the vocabulary with additional sub-triadic tokens (`0p`–`99p`). The meta-token explicitly indicates the original format. Since this approach slightly increases the sequence length, it is best reserved as an optional mode for domain-specific applications where original formatting is critical.

- **Alternative grouping sizes**: The choice of 3-digit groups (base-1000) is natural for human readability, but other sizes could be considered. Language-specific grouping (e.g., 3-2-2 in India, 4-4 in China) can be handled by the parser without changing the core vocabulary [12]. The decision should be guided by what representation is most efficient for the model.

- **Tokenisation variants**: Option B (compound tokens) produces shorter input sequences (lower *token count*) and gives the model ready-made magnitude-digit units, eliminating any ambiguity about which suffix belongs to which digit group. Option A (separate suffixes) adds only 10 new tokens. The trade-off between sequence length and vocabulary size remains an empirical question.

- **Comparison with xVal and NTL**: xVal demonstrates strong interpolation performance [11]; NTL offers a complementary training-level approach [9]. Their combination with TST may yield synergistic improvements.

7 Conclusion

We present Triadic Suffix Tokenization (TST), a scheme that transforms numerical tokenization from a source of errors into a structured representation. Unlike right-to-left commas, which only group digits, TST gives the model explicit knowledge of magnitude for integers and the same structural clarity for fractions. Our analysis compared TST with existing approaches, including digit-level tokenization, xVal, right-to-left commas, NumeroLogic, and NTL. While commas provide grouping without magnitude, and digit count prefixes improve accuracy on short numbers, none offer explicit per-triad magnitude annotation or deterministic boundary resolution. TST addresses these limitations through a fixed bijective mapping between suffixes and orders of magnitude, replicated markers for fractional depth, and implicit length information for integer parts.

The theoretical advantages of TST — explicit hierarchy, deterministic boundaries, and scalability across 33 orders of magnitude (10^{-15} to 10^{18}) — suggest that it may outperform existing methods across the full numeric range. While we focus on $N = 3$ for human readability, the scheme is fundamentally extensible to any group size N and can be expanded indefinitely by introducing additional suffix markers. By removing tokenization ambiguity and ensuring canonical representation through zero padding, the scheme ensures that the model receives an unambiguous gradient signal for numerical values, which may reduce both inference errors and training cost. The scheme requires no architectural modifications, only tokenization preprocessing, making it a practical drop-in enhancement. Moreover, TST is orthogonal to training-level improvements such as NTL; their combination may yield synergistic benefits.

Empirical validation on numerical reasoning benchmarks such as NumericBench [1] (comparing TST against digit-level tokenization, xVal [11], right-to-left commas [4, 5], and Nu-

meroLogic [8]) remains for future work. If confirmed, TST could offer a simple yet powerful enhancement to any language model that must reason about numbers.

References

[1] Li, H., Chen, X., Xu, Z., Li, D., Hu, N., Teng, F., Li, Y., Qiu, L., Zhang, C. J., Li, Q., & Chen, L. (2025). Exposing Numeracy Gaps: A Benchmark to Evaluate Fundamental Numerical Abilities in Large Language Models. *arXiv:2502.11075*.

[2] Daibasoglu, K. (2025). Probing the Sequential Enumeration Skills of Large Language Models. Master's thesis, Università di Padova.

[3] Yang, H., et al. (2024). Number Cookbook: Number Understanding of Language Models and How to Improve It. *arXiv:2411.03766*.

[4] Singh, A. K., & Strouse, D. (2024). Tokenization counts: the impact of tokenization on arithmetic in frontier LLMs. *arXiv:2402.14903*.

[5] Zhou, Z., et al. (2024). Scaling Behavior for Large Language Models regarding Numeral Systems: An Example using Pythia. In *Findings of EMNLP 2024*. *arXiv:2409.17391*.

[6] Loukas, E.-P., & Spyropoulou, E. (2025). System and Method for Automatically Tagging Documents. US Patent, IIT DICE.

[7] Kreitner, L., et al. (2025). Efficient numeracy in language models through single-token number embeddings. *arXiv:2510.06824*.

[8] Schwartz, E., Choshen, L., Shtok, J., Doveh, S., Karlinsky, L., & Arbelle, A. (2024). NumeroLogic: Number Encoding for Enhanced LLMs' Numerical Reasoning. *arXiv:2404.00459*.

[9] Zausinger, J., Pennig, L., Kozina, A., Sdahl, S., Sikora, J., Dendorfer, A., Kuznetsov, T., Hagog, M., Wiedemann, N., Chlodny, K., Limbach, V., Ketteler, A., Prein, T., Singh, V., Danziger, M., & Born, J. (2025). Regress, Don't Guess: A Regression-like Loss on Number Tokens for Language Models. In *Proceedings of the International Conference on Machine Learning (ICML 2025)*.

[10] Thawani, A., Pujara, J., & Kalyan, A. (2022). Estimating Numbers without Regression. In *NeurIPS Workshop on MATH-AI: Toward Human-Level Mathematical Reasoning*.

[11] Golkar, S., et al. (2023). xVal: A Continuous Number Encoding for Large Language Models. *arXiv:2310.02989*.

[12] O. Chetverina. (2026). *Triadic Suffix Tokenization: Reference Implementation and Vocabulary*. GitHub Repository. https://github.com/olgachetverina/triadic-suffix-tokenization