Are Non-BPE Tokenizers Ready to Improve LLM Numeracy?

Yizhan Huang [1] **Huang Nianchen** [2] **Yunxiang Li** [1] **Siu Po Wong** [1] **Michael R. Lyu** [1]

Abstract

Arithmetic remains a persistent weakness for large language models (LLMs). However, major research efforts have primarily focused on tool use or reasoning chains, which bypass this foundational limitation rather than addressing it directly. While tokenization has emerged as a promising line of research to tackle this numeracy issue, it still lacks a systematic evaluation. Regarding this gap, our paper asks whether alternatives to standard Byte-Pair Encoding (BPE) tokenization can improve arithmetic while preserving general natural-language (NL) ability.

We study three tokenizer innovations over a (fixed-vocabulary) BPE baseline: 1) *cross-tokenizer distillation* for controlled digit chunking, 2) *specialized number encodings* that represent numerical text with a <num> token, and 3) *byte-level LLMs* that operate on bytes instead of token-level. We build an evaluation protocol that requires LLM arithmetic capabilities to solve problems in both explicit and implicit ways. Surprisingly, our result shows that all three current alternatives could not outperform BPE in the arithmetic-NL Pareto frontier.

1. Introduction

Large Language Models (LLMs) are notoriously famous for poor numeracy. They keep struggling with arithmetic tasks such as the four basic operations [DLS+23, LCX+25, NRMB25]. For example, [YHK+25] reported that GPT-4o achieves an accuracy of only 5% on long-digit integer arithmetic.

Regarding poor numeracy, the main focus of research is on two strategies: 1) train LLMs to delegate the arithmetic computation to an *external tool* [GMZ+23, GSG+24, LWG+22, PZF22, SDYD+23]. 2) elicit LLMs to generate *reasoning chains* to decouple the problem into smaller steps [WWS+22, SLXK25, SWZ+24]. However, both methods are limited by efficiency: tool-use induces non-negligible system-level latency for end-users due to constructing and executing external API calls; reasoning chains may consume even thousands of tokens for one single calculation [KWB+25], sometimes reasoning chains are fragile [MAS+25]; In all, for both methods, the foundational limit of numeracy is ***bypassed*** instead of being directly addressed.

In addition to these approaches, tokenization remains an overshadowed yet promising direction for enhancing LLM numeracy. The tokenization community widely considers the dominant Byte-Pair Encoding (BPE) strategy [SHB16, KR18] to be a bottleneck for numerical tasks. By merging multiple characters into a single subword token, BPE obscures character-level information from the language model. As a result, LLMs fail to capture fine-grained numerical structures, leading to poor performance on numeracy benchmarks. Recently, the tokenization community has explored several alternatives: **cross-tokenizer distillation** (CTD) adapts a BPE vocabulary through training, enabling us to compare *n*-digit chunk tokenization [SS24] strategy under a fixed LLM; **specialized number encoding** proposes to encode numbers as a *separate* <num> token and process numbers in LLMs in a novel way instead of the traditional embedding matrix; **byte-level LLMs** adopt a UTF-8 byte as the vocabulary, aiming to boost character-level task performance and potentially benefit arithmetic tasks. A visualization is available in Figure 1.

Regarding LLMs on arithmetic tasks, one research gap lies in evaluation. It remains unclear whether LLMs trade off natural language (NL) capabilities for numeracy, given that current literature focuses purely on arithmetic evaluation [LCX+25, YHK+25]. Beyond that, another gap remains: no work has studied the potential of tokenization on arithmetic tasks. Among these innovations on tokenizers, some *implicitly* benefit LLM numeracy: cross-tokenizer distillation and byte-level LLMs are initially designed and evaluated on natural language tasks instead of numerical tasks. Some are designed *explicitly* to improve LLM numeracy: specialized number encodings process numerical text separately with NL tasks. But their performance on natural language tasks remains under-discussed. Existing numer-

[1]The Chinese University of Hong Kong [2]University of Southern California. Correspondence to: Yunxiang LI <yli@se.cuhk.edu.hk>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

Figure 1. Visualization of four tokenization strategies given an example string.

acy benchmarking work [LCX+25, YHK+25] also fails to explore tokenizer choices.

To address the gap, this paper aims to address the following research questions:

Do alternative tokenization strategies improve arithmetic capabilities without degrading general language ability?

We assessed all three methods (CTD, specialized number encodings, and byte-level LLMs), where the baseline is (fixed-vocabulary) BPE. Among these methods, CTD enables the adaptation of vocabulary within BPE, while specialized number encodings and byte-level LLMs explore non-BPE tokenization. Besides, we construct an evaluation dataset named TARITHEVAL (short for Tiered Arithmetic Evaluation) that holistically examines arithmetic performance when mixed with varying proportions of natural language.

As we will further discuss in § 4-6, through CTD, we demonstrate under a fixed LLM that smaller digit fragmentation in tokenization improves arithmetic performance, but hurts efficiency; specialized number encodings achieve impressive performance in arithmetic, but still fall short in NL performance compared with BPE; Byte-level LLM similarly fails to outperform BPE across both metrics.

In summary, our contribution is as follows:

1. To the best of our knowledge, we are the first to review tokenizer innovations in the context of arithmetic tasks in a systematic way.
2. Through empirical experiments, we verify that none of the recent methods stably outperforms BPE.
3. We discuss possible future directions of tokenization based on our experimental results.

2. Background

2.1. Tokenizer and Embedding

Tokenizer. We define the input alphabet as the set of bytes $\Sigma = \{0, \ldots, 255\}$, so that any text is a finite byte sequence $x \in \Sigma^\star$[1] under the UTF-8 encoding [Yer03]. A tokenizer is a tuple $(\mathcal{V}, T)$ comprising a finite vocabulary $\mathcal{V} \subset \Sigma^\star$ and a tokenization function $T : \Sigma^\star \to \mathcal{V}^\star$ that maps a byte string x to a token sequence $T(x) = (t_1, \ldots, t_n) \in \mathcal{V}^\star$.

Byte-Pair Encoding. The dominant choice for $(\mathcal{V}, T)$ in modern LLMs is *Byte-Pair Encoding* (BPE) [Gag94, KR18]. BPE constructs $\mathcal{V}$ by a greedy, data-driven merge procedure: starting from the single-byte vocabulary Σ, it iteratively identifies the most frequent adjacent token pair in the training corpus and adds their concatenation as a new token. The final vocabulary size $|\mathcal{V}|$, is a hyperparameter set by the model designer.

Embedding. Given $T(x) = (t_1, \ldots, t_n)$, the next step is to map token IDs through the embedding matrix. Fix an index map $\iota : \mathcal{V} \to \{1, \ldots, |\mathcal{V}|\}$ and an embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$. Let $\mathbf{o}_i$ be the one-hot row vector with its nonzero entry at $\iota(t_i)$; then lookup is exactly the matrix multiplication $\mathbf{x}_i = \mathbf{o}_i \mathbf{E} = \mathbf{E}_{\iota(t_i)}$. The embedding se-

[1] $\Sigma^\star$ denotes the Kleene closure of Σ, i.e., the set of all finite-length sequences (including the empty sequence) over the alphabet Σ.

quence $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is then passed to the main Transformer components, which produce hidden states $\mathbf{h}_i \in \mathbb{R}^d$.

Therefore, tokenizer adaptation from $(\mathcal{V}, T)$ to $(\mathcal{V}', T')$ is challenging since a new embedding matrix $\mathbf{E}' \in \mathbb{R}^{|\mathcal{V}'| \times d}$ is required. Adapting $\mathbf{E}$ to $\mathbf{E}'$ is non-trivial and typically requires re-training.

2.2. Beyond BPE: Emerging Tokenization Strategies for Numbers

Cross Tokenizer Distillation (CTD) Cross-tokenizer distillation adapts an LLM tokenizer without training a new model from scratch. The starting point is a pretrained *teacher* model with tokenizer $(\mathcal{V}, T)$. We then construct a *student* model with a target tokenizer $(\mathcal{V}', T')$ and train the student to imitate the teacher on the same underlying byte string x.

Teacher and student logits are defined over non-isomorphic vocabularies, the standard KL divergence is ill-posed. CTD methods therefore align distributions through shared byte spans or tokenizer-agnostic surrogates, as done by Universal Logit Distillation (ULD) or approximate likelihood matching (ALM) [BHHC25, MVP25].

In the context of arithmetics, CTD provides a controlled way to *test number-specific BPE variants*: we can remove or retain multi-digit numeric tokens, and adapt the resulting student from the original model instead of pretraining a separate model for each digit granularity.

Specialized Number Encoding Specialized number encodings process numbers as a separate `<num>` token rather than as a typical BPE token, while keeping BPE for other forms of text. The methods typically adapt an existing LLM with an additional LLM head for number encoding / decoding. For example, FoNE uses Fourier features to represent numerical values [ZFS+26]. This design avoids the arbitrary segmentation of multi-digit numbers and gives the model more direct access to magnitude, but it also changes the input representation itself and typically requires training the model with this encoding from the start. We therefore view specialized number encoding as an explicitly numeracy-oriented alternative to general-purpose tokenization.

Byte-level LLMs In byte-level LLMs, the vocabulary is the byte alphabet itself: $\mathcal{V} = \Sigma$, so $|\mathcal{V}| = 256$, and T maps each UTF-8 byte to one token. For decimal numerals, this representation exposes every digit and symbol directly. For byte transformers, the main obstacle is sequence length: raw byte streams are much longer than subword token sequences. To reduce the cost of running a full Transformer over every byte, most recent byte-level LLMs use a hierarchical or "latent transformer" design [PPR+25, NDEB25, HWG25, MML+25]. A lightweight local encoder first contextualizes byte embeddings; a boundary mechanism pools adjacent bytes into latent patches; a deep global Transformer operates over this shorter patch sequence, and a local decoder maps patch-level states back to next-byte predictions.

Byte-level LLMs are a natural candidate for numeracy, although existing byte-level models are usually additionally trained and evaluated on character-level understanding corpus, where arithmetic data is rarely explored [MML+25, PPR+25].

Tier	Example	Source
1	`36 + 12 =` $\rightarrow$ `48`	Synthetic
2	*What is the min of* $[3, 7, 2, 9, 1]$*?* $\rightarrow$ `1`	NUMERICBENCH
3	*Natalia sold 48 clips in April and half as many in May. How many altogether?* $\rightarrow$ `72`	GSM8K, MULTIARITH

Table 1. Summary of the three arithmetic evaluation tiers.

3. Evaluation Protocol

To holistically assess LLM performance with new tokenizers, our protocol covers two dimensions: (i) *arithmetic*. Unlike previous work, we design a *three-tier* suite in which the share of natural-language content grows monotonically from Tier 1 to Tier 3. We name our dataset as **TARITHEVAL**, where `TArith` is short for "Tiered Arithmetic". (ii) *general natural language*, where tasks are dominated by natural language content. We restrict our evaluation suite to pre-trained ("base") models. This is because LLMs at byte-level or with specialized mathematical encodings are predominantly base models. Almost no official instruction-tuned checkpoints for these architectures are currently released.

We report a scalar score for each dimension. For arithmetic tasks, it is the macro-mean of tiers 1-3, i.e., $\frac{1}{3}(\mu_{T_1} + \mu_{T_2} + \mu_{T_3})$; where μ_{T_k} is the mean accuracy across the tasks in Tier k. On the natural language dimension, we report the average of all composite datasets.

Arithmetic Evaluation. We describe the composition of TARITHEVAL here. *Tier 1 basic four operations*. Each instance is a single expression of the form `a op b` with `op` $\in \{$`+`, `-`, `*`, `/`$\}$. The default operand ranges from 1 to 5. Beyond that, for some experiments, we include an additional evaluation with operand range from 6-9 to evaluate length generalization abilities [SS24]. *Tier 2 Fixed Natural Language Templates.* This tier places integer-list operations inside fixed natural-language templates. We select tasks from $\{\min, \max, \text{sum}, \text{count}, \text{sort}\}$. *Tier 3 Free Form Natural-Language Arithmetic.* This tier evaluates LLMs using free-form natural language math word problems, where the underlying computations remain basic arithmetic opera-

tions. Compared to Tier 2, the natural language context here is significantly more diverse and constitutes a larger proportion of the prompt. Both tier 2 and 3 require an implicit arithmetic calculation to complete the task.

As a summary, Table 1 gives examples of each tier.

Natural Language (NL) Evaluation. The NL dimension evaluates general language ability, where almost no numerical text is involved. We adopt a multiple-choice style evaluation. Datasets include MMLU [HBB+21], ARC-CHALLENGE [CCE+18], and BOOLQ [CLC+19].

4. Cross Tokenizer Distillation (CTD)

4.1. Motivation

In this section, through cross tokenizer distillation (CTD), we explore ***n*-digit chunking**, the BPE schema that partitions a numeric sequence into blocks of at most n digits. For instance, under 1-digit-chunking ($n = 1$), `12345` is tokenized to `1|2|3|4|5`; similarly for $n = 3$, the same string would be tokenized to `123|45`. In fact, currently different frontier LLMs adopt different n to chunk digits. For example, GPT-4 [Ope23] and Llama 3 [DJP+24] family caps digit chunking to at most three digits ($n = 3$), while Gemma family [GG24] uses only one-digit tokens (i.e., $n = 1$).

The design n is decided based on empirical experiences of LLM trainers, however, the effect of n on LLM numeracy is not clear. Therefore, with CTD, we adapt n for n-digit-chunking for a fixed LLM. Specifically, we start from native vocabulary as the teacher configuration $(\mathcal{V}, T)$ as our $n = 3$ baseline. We then apply CTD to adapt the student $(\mathcal{V}', T')$ to constrained chunking boundaries where $n \in \{1, 2\}$, thus isolating the exact trade-off between numerical accuracy and decoding cost.

4.2. Distillation Methodology and Framework

We implement CTD using the Generalized Logit-Distillation (GOLD) trainer [vWBT+20]. The student is optimized by a composite loss balancing two equally weighted signals: the Universal Logit Distillation (ULD) [BHHC25] loss to bridge the vocabulary mismatch, and a standard supervised cross-entropy loss on reference target completions to stabilize training. The core ULD component solves the vocabulary mismatch problem in distillation, by aligning the two models using a loss function over the sorted probability mass across shared token sequence rather than individual token slots.

We evaluate our framework across model bases whose native vocabularies inherently function as $n = 3$ configurations. Specifically, we select three base models spanning an order of magnitude in scale: OLMo-3-7B, Llama-3.2-1B, and OLMo-2-1B.[2] For each base, we vary the maximum chunk size $n \in \{1, 2\}$ against the native $n = 3$ baseline, isolating chunking mechanics from pretraining confounds.

Students are trained on 60K elementary arithmetic problems from TARITHTRAIN, and hyperparameter settings deferred to Table 6. To isolate tokenization changes from simple task exposure, we introduce an *in-tokenizer SFT baseline* that fine-tunes the identical base models using their unmodified native vocabularies. Evaluation follows the tiered protocol in Section 3; to capture the efficiency trade-off, we additionally track decoding cost, quantified by the average number of generated tokens per example. We additionally trained the off-policy variant ($\lambda = 0$); the resulting trade-offs, together with supervised fine-tuning result, are given in Figure 5 in Appendix D.

4.3. CTD Results

Following Section 3, we use TARITHEVAL to evaluate CTD across three core metrics: arithmetic accuracy, general natural language (NL) ability, and computational efficiency (measured via output token count). Figure 2 plots accuracy against average decoded tokens, revealing three key insights.

Smaller chunks deliver large arithmetic gains. On in-distribution arithmetic (Tier 1), reducing chunk size yields consistent, scale-invariant gains: shifting from $n = 3$ to $n = 1$ boosts accuracy from 27% to 83% (OLMo-3-7B), 14% to 74% (Llama-3.2-1B), and 4% to 61% (OLMo-2-1B). This uniform 55–60 point leap indicates that explicit numerical representation, rather than model scale, is the binding constraint. Length generalization (extending operand lengths to 6–9 digits as described in Section 3) exhibits this pattern even more starkly: native $n = 3$ tokenization collapses due to novel token merges on longer operands, whereas $n = 1$ enables robust extrapolation by maintaining a stable, digit-by-digit vocabulary.

These gains cost substantially longer outputs. These accuracy breakthroughs incur a direct efficiency penalty: $n = 1$ students decode roughly 3.9 tokens per example—a 1.9–2.6$\times$ inflation over baselines that causes the rightward shift in Figure 2.

Natural language accuracy remains essentially unchanged across all settings. Crucially, downstream NL benchmark performance remains flat across all configurations. This stability confirms that our distillation framework achieves targeted numeric adaptation without eroding the

[2] OLMo-3-7B and OLMo-2-1B refer to `allenai/Olmo-3-1025-7B` and `allenai/OLMo-2-0425-1B`, respectively.

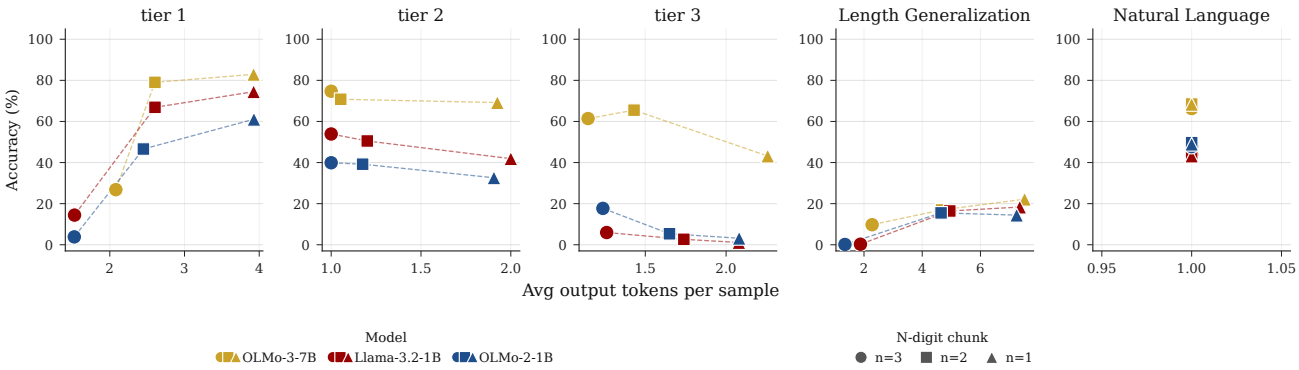

Figure 2. Arithmetic accuracy versus average decoded tokens across model bases and task tiers. Each curve traces the performance trade-off from coarse-grained ($n = 3$) to fine-grained ($n = 1$) tokenization.

model's underlying linguistic competence.

Additionally, we conduct an ablation study, where we only do a native SFT training on the same dataset. We report the results in Figure 5 in Appendix D.

> It is verified that n-digit tokenization has a negligible impact on NL performance; A smaller n enhances arithmetic capability, but LLMs trade this for computational efficiency.

5. Specialized Number Encodings

In this section, we employ two representative methods to encode numbers: FoNE [ZFS+26] and BitToken [KHM+25]. As a recall of Section 2.2, both replace each numeric span with a single `<num>` token and pass the parsed numeric value to the model together with the token ID. Specifically, FoNE embeds the value using base-10 Fourier features, whereas BitToken embeds its IEEE-754 float64 bit representation together with a reciprocal feature; in both cases, the resulting value embedding is injected before the transformer. Both methods require training from *scratch* due to the special design of tokenizers.

Note that both papers only evaluate perplexity of a held-out pre-training corpus to report NL result, which is limited. We therefore train each model from scratch and evaluate whether explicit number encodings improve arithmetic without degrading natural-language performance.

5.1. Experimental Setup

We follow the BitToken codebase and data artifact for training. The training mixture combines a 10B pre-training fineweb [PKa+24] corpus, and around 0.2B synthetic arithmetic data, including various sub-tasks.

We compare FoNE, BitToken, and a BPE baseline under a matched training setup. We adopted an LLM architecture based on an improved version of nanoGPT [JBR+24] in 114M size. FoNE and BitToken use the single-token-number implementation, passing both token IDs and numeric values to the model; while the BPE baseline uses the embedding matrix in the standard way. We defer optimizer, curriculum, and data-processing details to Appendix E.

5.2. Experimental Results

Table 2. Specialized number encodings versus the BPE baseline. Arithmetic average is the macro-average over Tiers 1–3.

Metric	FoNE	BitToken	BPE
Tier 1	78.54	**95.75**	86.79
Tier 2	10.30	7.80	**12.20**
Tier 3	**2.53**	1.56	1.82
Arith. avg	30.46	**35.04**	33.60
NL avg	27.41	28.63	**31.16**

We evaluate three pre-trained models (FoNE, BitToken, and BPE) on TARITHEVAL and natural language following the setup established in Section 3. Table 2 shows that specialized number encodings do not uniformly dominate the BPE baseline. BitToken is strongest on direct arithmetic, improving Tier 1 and the arithmetic macro-average, but it lags the BPE baseline on Tier 2 and NL. FoNE obtains the best Tier 3 score, yet its overall arithmetic average remains below BitToken and BPE. Thus, in this scratch-trained setting, explicit number encodings help some numeric regimes but do not produce a Pareto improvement across arithmetic and general language.

> While specialized number encodings achieve impressive performance on arithmetic, especially in tier 1, they still fall short in NL performance compared with BPE.

6. Byte-level LLM

Byte-level LLMs replace subword units with UTF-8 bytes, exposing digits and arithmetic symbols directly to the model. We study this design in two steps: first, we evaluate public byte-level checkpoints against their BPE counterparts on the arithmetic–NL frontier; second, motivated by the limited arithmetic supervision in existing byte-level training recipes, we fine-tune Bolmo on TARITHTRAIN and compare its gains with the corresponding subword Olmo model.

6.1. Evaluating public byte-level models

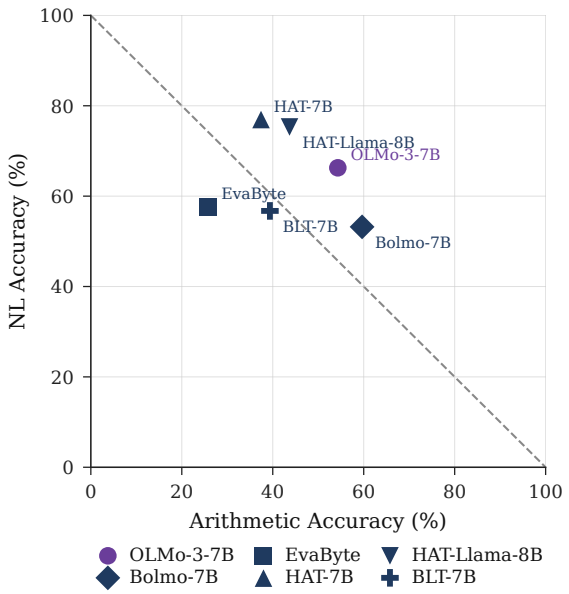

Figure 3. Comparison between BPE and byte-level tokenization for the 7B models.

Following the evaluation protocol in §3, we evaluate recent public byte-level families, including EvaByte [ZZW+25], HAT [NDEB25], BLT [PPR+25], and Bolmo [MML+25], together with the BPE-based Olmo [EBK+25]. To ensure a fair comparison, although some byte LLMs have instruction-tuned variants, we decide not to include them. Figure 3 focuses on 7B-or-larger checkpoints, with the 1B comparison deferred to Appendix Figure 6.

We find that these public checkpoints lie near the same empirical arithmetic–NL frontier rather than showing a clear byte-level advantage. Bolmo and Olmo are comparable because they share the same backbone, with Bolmo obtained by byteifying Olmo; even for this comparison, the observed trade-off appears coupled to the training recipe, including Bolmo's additional character-level supervision. This motivates the controlled fine-tuning comparison below.

6.2. Fine-tuning Byte-Level LLMs on Arithmetic Tasks

Motivation We observe that arithmetic data is largely absent from the training mixes of existing byte-level LLMs. Bolmo [MML+25] is a representative case: when byteifying Olmo, the authors augment the pretraining data mix with character-level understanding tasks (e.g., CUTE-style [ESF24] spelling and character manipulation) but include no arithmetic-related supervision. We therefore suspect that the weak arithmetic performance of current byte-level LLMs stems in part from this gap, and fine-tuning Bolmo on arithmetic data may *potentially* improve its capabilities in this regard.

In this section, we fine-tune Bolmo 7B on a curated dataset named TARITHTRAIN, and compare its gains against subword counterpart Olmo-3-7B.

Table 3. Composition of the TARITHTRAIN fine-tuning mix (100,000 examples).

Tier	Share	Source
1 Pure Arith.	60%	Synthetic
2 Templated Arith.	20%	NumeracyBench
3 Free Form Arith.	10%	GSM8k
NL	10%	tulu-3-sft-mixture
Total	100%	

Fine-tuning Dataset Following the tiered structure of TARITHEVAL, we construct **TARITHTRAIN** as a supervised fine-tuning dataset. It covers all three arithmetic tiers and the NL tier, but allocates the largest share to Tier 1 pure arithmetic because the intervention targets basic arithmetic competence. Table 3 summarizes the mixture; Appendix F provides the construction details.

Baseline Comparison We fine-tune both Bolmo 7B and Olmo-3-7B on the same dataset. Because Bolmo and Olmo use different tokenization strategies, the resulting token distributions differ substantially. We therefore start from the same experimental recipe for both models and run up to ten rounds of hyperparameter sweeping to avoid training instability and over/under-fitting. The selected configurations are recorded in Appendix F.2.

Experimental Results Figure 4 reports arithmetic and NL accuracy for the two base models and their TARITHTRAIN-fine-tuned variants. In general, we observe that the subword baseline benefits substantially from the same SFT recipe: Olmo-3-7B improves arithmetic capability over (+12.93pp), while NL accuracy is essentially unchanged. In contrast, the byte-level model gains far less from the identical mixture: Bolmo-7B gains less than 1% on arithmetic tasks, but with

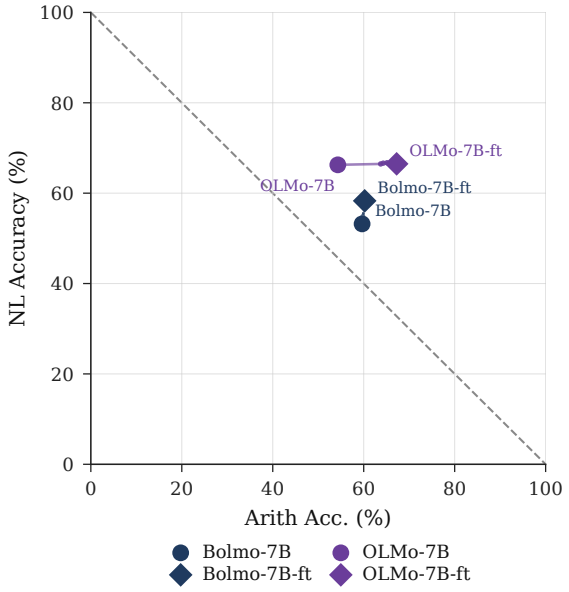

Figure 4. Arithmetic accuracy vs. natural-language accuracy before and after fine-tuning on TARITHTRAIN. diamonds denote fine-tuned variants; arrows trace each model's pre/post-SFT shift.

a clearer NL improvement ($53\% \rightarrow 58\%$). To summarize, Olmo dominates Bolmo on both arithmetic and natural-language tasks after fine-tuning.

> Without fine-tuning, frontier byte-level and BPE LLMs lie on nearly the same frontier; after fine-tuning, however, BPE-based LLMs clearly dominate their byte-level counterparts.

7. Related Work

7.1. Numeracy Benchmarking

[LCX+25] introduce NUMERICBENCH, a suite of six template-based skills: number recognition, arithmetic, contextual retrieval, comparison, summary, and multi-step reasoning. [YHK+25] introduce NUPA, with 17 numerical tasks across 4 representations, and find that techniques designed to improve numerical ability do not transfer to pre-trained models. All of these benchmarks evaluate already-trained models on a fixed task and language format, and none of them probe how the tokenization strategy itself affects performance.

7.2. Methods To Improve LLM Numeracy

Reasoning chain for math One line of methods improves mathematical reasoning by making solution traces explicit. [WWS+22, YYZ+23] show that chain-of-thought prompts let a model write out intermediate steps and improve math accuracy. [WWS+23] sample many such chains and take the majority vote. Beyond prompting, [SWZ+24] introduces GRPO to train LLMs to elicit reasoning chains.

Tool use for math Another line trains the LLM to use an external solver for numerical problems. PAL [GMZ+23] and TORA [GSG+24] write code and run it instead of predicting the number directly. [HYPWG23] pair the language model with a symbolic solver, and [SDYD+23] train the model to call tools in context. Tool use moves the arithmetic outside the model, so the tokenizer-level question we ask does not apply.

Our experiments are orthogonal to these approaches: we evaluate whether tokenization itself affects arithmetic ability, without adding reasoning chains or tool calls. We refer readers to Appendix G for additional related work.

8. Conclusion and Future Work

In this paper, we systematically study whether tokenizer alternatives can improve LLM arithmetic without sacrificing general natural-language ability. Across cross-tokenizer distillation, specialized number encodings, and byte-level LLMs, our results suggest that no single alternative consistently outperforms traditional Byte-Pair Encoding (BPE) across both dimensions. These findings inspire several directions for future work.

Cross-tokenizer distillation: generalize beyond BPE. A central limitation of current CTD methods is that they remain designed for BPE-based LLMs. These methods operate on *logit* level: the next-token distribution that the model produces through its (*BPE-native*) output-embedding matrix. As a result, they can transfer only the knowledge that the teacher exposes through logit. But in fact, non-BPE tokenizers use a different LM head other than logit, hence some information may remain unused during knowledge distillation. For example, BitToken illustrates where this assumption breaks down. In such models, the numerical value is not read from the token logits; instead, it is decoded by a separate head attached to the final hidden state. The signal needed for transfer therefore lies outside the token-logit channel that existing CTD methods align, and recovering it is non-trivial. We nonetheless foresee that addressing this problem would substantially broaden the reach of CTD, and hence boost LLM numeracy.

Specialized number encodings: scaling models and advanced routing. A promising direction is to move from hard-coded numerical routing to learned, context-dependent routing. Current number encodings detect numbers with a fixed rule, and route to a pre-determined computation graph based on heuristics. Following the same philosophy

of "handling NL and numerical text separately", we envision that both number "detection" and computation graph could be learned in an end-to-end way with low supervision instead of hard-coded. Besides, we should train and evaluate these LLMs at a larger scale, whether larger models break arithmetic-NL trade-off is not clear.

Byte-level LLMs: Build a stronger ecosystem. The byte-level LLM ecosystem remains less mature than the standard subword-model ecosystem in both engineering and modeling terms. On the engineering side, many evaluation, serving, and optimization libraries assume `transformers`-compatible subword tokenizers and model interfaces. Most byte-level implementations therefore cannot directly enable batching utilities or reuse existing optimization libraries (for BPE LLMs). Training and inference are therefore limited by the current infrastructure.

On the modeling side, byte-level LLMs may "absorb" capabilities from strong BPE-based LLMs. Recent work [MML+25] suggests that byte-level capabilities can be introduced through model merging [IRW+23], pointing to a path in which byte-level LLMs acquire advanced capabilities under a very low compute budget. We view this as a promising but still underexplored direction, especially at larger scales and for byte-level models whose behavior has been shaped by instruction tuning or RL.

Limitations

Our paper is limited in two aspects.

Model scale. Our experiments are constrained to small- and medium-scale open models, rather than frontier-scale systems. In specialized number encoding (Section 5), our experiments work on a 114M-parameter nanoGPT-2 model, since full pre-training is required. This is mainly limited by compute.

Evaluation in floating point. In this paper, we focus our evaluation on integer arithmetic. Investigating integers alone serves as a sufficient differentiator among the various tokenization methods; theoretically, these findings can be extended to floating-point numbers as well.

References

[ALP+25] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, et al. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025.

[BHHC25] Nicolas Boizard, Kevin El Haddad, CELINE HUDELOT, and Pierre Colombo. Towards cross-tokenizer distillation: the universal logit distillation loss for LLMs. *Transactions on Machine Learning Research*, 2025.

[BvGO25] Tanja Baeumel, Josef van Genabith, and Simon Ostermann. The lookahead limitation: Why multi-operand addition is hard for llms. *CoRR*, abs/2502.19981, 2025.

[CCE+18] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[Cha24] François Charton. Learning the greatest common divisor: explaining transformer predictions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[CKB+21] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021.

[CLC+19] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pages 2924–2936, 2019.

[DJP+24] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Ashutosh Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mahajan, Aman Banbury, Alejandro Alcarde Seoane, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[DLS+23] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. Faith and

fate: Limits of transformers on compositionality. 2023.

[EBK+25] Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee F. Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, William Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3. *CoRR*, abs/2512.13961, 2025.

[ESF24] Lukas Edman, Helmut Schmid, and Alexander Fraser. Cute: Measuring llms' understanding of their tokens. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3017–3026, 2024.

[Gag94] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, February 1994.

[GG24] Gemma Team and Google. Gemma 2: Improving open language models via better size-transfer and modern architectures. *arXiv preprint arXiv:2408.00118*, 2024.

[GMZ+23] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pages 10764–10799. PMLR, 2023.

[GSG+24] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[GTA+24] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024.

[HBB+21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[HWG25] Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. *arXiv preprint arXiv:2507.07955*, 2025.

[HYPWG23] Joy He-Yueya, Gabriel Poesia, Rose Wang, and Noah Goodman. Solving math word problems by combining language models with symbolic solvers. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

[IRW+23] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023.

[JBR+24] Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024.

[KHM+25] Linus Kreitner, Paul Hager, Jonathan Mengedoht, Georgios Kaissis, Daniel Rueckert, and Martin J Menten. Efficient numeracy in language models through single-token number embeddings. *arXiv preprint arXiv:2510.06824*, 2025.

[KPR+23] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[KR18] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[KWB+25] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. In *Advances in Neural Information Processing Systems*, volume 38, pages 92213–92266, 2025.

[LCX+25] Haoyang Li, Xuejia Chen, Zhanchao Xu, Darian Li, Nicole Hu, Fei Teng, Yiming Li, Luyu Qiu, Chen Jason Zhang, Qing Li, and Lei Chen. Exposing numeracy gaps: A benchmark to evaluate fundamental numerical abilities in large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, Findings of ACL, pages 20004–20026. Association for Computational Linguistics, 2025.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[LJL+24] Kaifeng Lyu, Jikai Jin, Zhiyuan Li, Simon Shaolei Du, Jason D. Lee, and Wei Hu. Dichotomy of early and late phase implicit biases can provably induce grokking. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[LMP+25] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025.

[LWG+22] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022.

[MAS+25] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[MBS+24] Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. Transformers can do arithmetic with the right embeddings. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: An-*

nual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024.

[MML+25] Benjamin Minixhofer, Tyler Murray, Tomasz Limisiewicz, Anna Korhonen, Luke Zettlemoyer, Noah A. Smith, Edoardo M. Ponti, Luca Soldaini, and Valentin Hofmann. Bolmo: Byteifying the next generation of language models, 2025.

[MVP25] Benjamin Minixhofer, Ivan Vulic, and Edoardo Maria Ponti. Cross-tokenizer distillation via approximate likelihood matching. *CoRR*, abs/2503.20083, 2025.

[NCL+23] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[NDEB25] Pit Neitemeier, Björn Deiseroth, Constantin Eichenberg, and Lukas Balles. Hierarchical autoregressive transformers: Combining byte-and word-level processing for robust, adaptable language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[NRMB25] Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*, 2025.

[Ope23] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[PKa+24] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[PPR+25] Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodríguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E. Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 9238–9258. Association for Computational Linguistics, 2025.

[PZF22] Aaron Parisi, Yao Zhao, and Noah Fiedel. TALM: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

[RR15] Subhro Roy and Dan Roth. Solving general arithmetic word problems. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752. The Association for Computational Linguistics, 2015.

[SDYD+23] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[SHB16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[SLXK25] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[SS24] Aaditya K. Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier llms, 2024.

[SWZ+24] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[vWBT+20] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020.

[WWS+22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[WWS+23] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[Yer03] Francois Yergeau. Utf-8, a transformation format of ISO 10646. *RFC*, 3629:1–14, 2003.

[YHK+25] Haotong Yang, Yi Hu, Shijia Kang, Zhouchen Lin, and Muhan Zhang. Number cookbook: Number understanding of language models and how to improve it. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[YYZ+23] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36:11809–11822, 2023.

[ZFS+26] Tianyi Zhou, Deqing Fu, Mahdi Soltanolkotabi, Robin Jia, and Vatsal Sharan. FoNE: Precise single-token number embeddings via fourier features. In *The Fourteenth International Conference on Learning Representations*, 2026.

[ZZW+25] Lin Zheng, Xueliang Zhao, Guangtao Wang, Chen Wu, David Dong, Angela Wang, Mingran Wang, Yun Du, Haige Bo, Amol Sharma, Bo Li, Kejie Zhang, Changran Hu, Urmish Thakker, and Lingpeng Kong. Evabyte: Efficient byte-level language models at scale, 2025.

A. Compute

All experiments were conducted on a Linux server with four NVIDIA A800 GPUs.

B. Ethics Statement

The authors foresee no adverse ethical implications arising from this work. All models, training data, and evaluation benchmarks are sourced from publicly released sources. Furthermore, the authors have ensured that all datasets utilized comply strictly with their intended uses and license agreements.

C. Additional Evaluation Protocol Details

Tables 4 and 5 summarize the complete evaluation protocol. Table 4 details the arithmetic suite used in TARITHEVAL, including the tier structure, split size, prompting regime, source citation, and license; Table 5 lists the natural-language controls used to measure whether tokenizer changes preserve general language understanding. We use greedy decoding ($\text{temperature} = 0$) and follow the n-shot setting listed for each task. Arithmetic generations are scored by exact match after answer extraction with the LM Evaluation Harness output-processing framework and task-specific regular expressions [GTA+24]: Tier 1 and Tier 2 use the first integer in the model response, while Tier 3 uses the final integer or boxed-answer pattern, depending on the dataset wrapper.

D. Additional Cross-Tokenizer Distillation Details

Table 6 lists the hyperparameters for our Cross-Tokenizer Distillation (CTD) framework, which are kept identical across all three base models. The distillation objective combines ULD and GKD with equal weights ($1.0/1.0$) and an interpolation factor of $\beta = 0.5$. We set the mixing parameter to $\lambda = 0.5$, allowing the student to train on a 50/50 mix of static dataset sequences and its own self-generated completions. Low-Rank Adaptation (LoRA) is applied to all linear projections within the attention and MLP layers.

To evaluate the necessity of on-policy mixing, we conduct an ablation experiment under a pure off-policy regime by setting $\lambda = 0$. Here, the student trains exclusively on the fixed text trajectories of the arithmetic dataset while matching the teacher's logit distribution.

Tier	Dataset	Breakdown	Used size	Source	License
1	Synthetic four-operation arithmetic	Synthetic integer expressions over addition, subtraction, multiplication, and exact division; operands have 1–5 digits. We also use an additional 6–9 digit version for length generalization.	8,000 test examples for the default set (2,000 per operation); 8,000 additional test examples for length generalization. 0-shot.	Synthetic dataset	-
2	Synthetic list arithmetic	Fixed-template list operations: minimum, maximum, sum, count, and sorted-position lookup. Each problem uses a length-5 list of two-digit integers. The tasks are adapted from the `num_list` style of NUMERICBENCH.	1,000 test examples (200 per operation). 5-shot prompting; few-shot examples are drawn from the test split with evaluation-document filtering.	NUMERICBENCH [LCX+25]; generated in-house for the integer-only setting.	MIT
3	GSM8K `main` and MULTIARITH	Free-form grade-school word problems with arithmetic embedded in natural language. GSM8K is evaluated with the boxed-answer wrapper; MULTIARITH is evaluated with a plain question-answer wrapper.	1,319 GSM8K test examples and 180 MULTIARITH test examples. Both use 5-shot prompting from their training splits.	GSM8K [CKB+21]; MULTIARITH [RR15].	GSM8K: MIT; MULTIARITH: Apache 2.0

Table 4. Detailed breakdown of the arithmetic evaluation data.

Dataset	Breakdown	Used size	License
MMLU [HBB+21]	Massive multitask multiple-choice questions across 57 subjects.	14,042 test examples; 5-shot prompting.	MIT
ARC-CHALLENGE [CCE+18]	Grade-school science multiple-choice questions from the challenge split.	1,172 test examples; 25-shot prompting.	CC BY-SA 4.0
BOOLQ [CLC+19]	Binary yes/no reading-comprehension questions over short passages.	3,270 validation examples; 0-shot prompting.	CC BY-SA 3.0

Table 5. Detailed breakdown of the natural-language evaluation data.

Figure 5 evaluates distillation at $\lambda = 0$ against native-tokenizer SFT controls ($\star$). The results show that while native SFT successfully recovers in-distribution arithmetic at a lower decoding cost, it completely fails at length generalization. Crucially, without on-policy mixing ($\lambda = 0$), the digit-chunked students ($n \in \{1, 2\}$) also suffer severe degradation across all tiers. This proves that mixed sampling ($\lambda = 0.5$) is required to achieve the robust generalization advantages of digit-level tokenization.

E. Additional Number Encoding Details

Table 7 summarizes the implementation choices for the specialized number-encoding experiments in Section 5.

Training mixture. The natural-language component is drawn from FineWeb [PKa+24]. The synthetic arithmetic component is generated at 30M training examples per sub-task. The main decimal setting uses six sub-tasks—addition, multiplication, division, minimum/maximum, interval classification, and sorting—for 180M synthetic arithmetic examples in total. The BitToken setting keeps the same 30M-example scale per sub-task, but uses binary multiplication and binary division-modulo as curriculum variants, with decimal multiplication and decimal division reserved for the endgame curriculum. Thus, the active arithmetic training pool is approximately 0.18B examples, which we round to about 0.2B in the main text.

Optimization. All runs train from scratch for a 10B-token budget using `bfloat16`, effective batch size 192, learning rate 0.02, Muon optimization, and cosine decay after a 10% warmup. Curriculum sampling is enabled for the arithmetic tasks whose difficulty can be ordered, and the number-token models add the auxiliary numeric loss with weight 10.

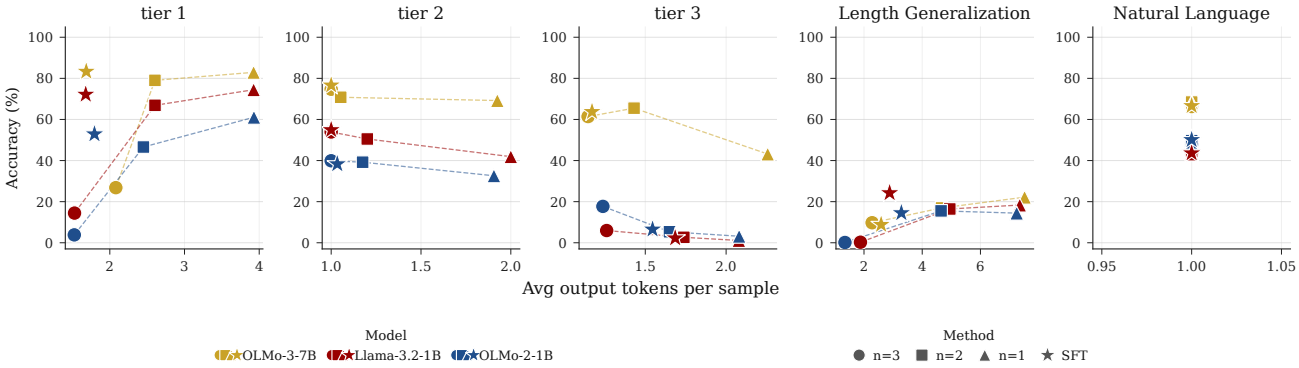

Figure 5. Accuracy versus mean output length under off-policy distillation ($\lambda = 0$), with supervised fine-tuning (SFT) baselines. Each panel corresponds to one task tier. Colour denotes the base model and marker denotes the training regime: n=3 is the untuned teacher, n=1 and n=2 are digit-chunked distillation, and $\star$ is SFT on the model's native tokenizer. Each point is the mean accuracy over the tasks in a tier; the horizontal axis is the mean number of generated tokens per example.

Table 6. Cross-tokenizer distillation hyperparameters. The same recipe is used for all three bases.

Hyperparameter	Value
Base / teacher models	OLMo-3-7B Llama-3.2-1B OLMo-2-1B
Distillation objective	GOLD + ULD (hybrid)
ULD CE / KD weight	1.0 / 1.0
KD interpolation β	0.5
On-policy mixing λ	0.5 (on-policy)
Student / teacher temperature	1.0 / 1.0
PEFT	LoRA (r=16, α=32)
LoRA dropout	0.05
LoRA targets	all linear proj. (attn + MLP)
Learning rate	1×10^{-4}
LR schedule	linear, no warmup
Optimizer	Adam
Max gradient norm	1
Effective batch size	16 (2×8 grad. accum.)
Max sequence length	32
Precision	bf16 (TF32 matmul)

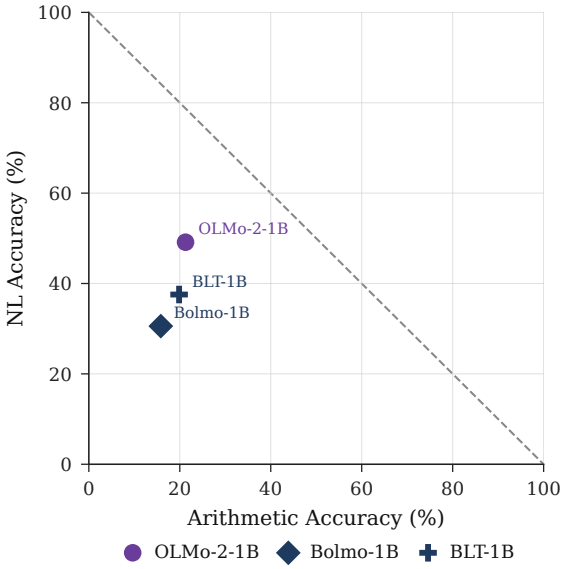

Figure 6. Comparison between BPE and byte-level tokenization for the 1B models.

F. Additional Material on Byte-Level LLMs

F.1. TARITHTRAIN Details

TARITHTRAIN is a 100,000-example supervised fine-tuning mix designed to mirror the tiered structure of Appendix C while placing most of the training mass on direct arithmetic. Each example is a JSON-style prompt–completion pair with a source tag. We generate the mix with random seed 42 and apply one final shuffle before training.

Pure arithmetic. The largest component contains 60,000 synthetic examples covering the four integer operations: 15,000 addition, 15,000 subtraction, 15,000 multiplication, and 15,000 exact-division rows. Prompts use the compact form `a+b=`, `a-b=`, `a*b=`, or `a/b=`, and completions contain only the integer result. Operands are sampled with one to five digits. For subtraction, examples are constrained to have nonnegative integer answers; for division, examples are generated by first sampling a divisor and integer quotient so that the dividend is exactly divisible.

Templated list arithmetic. The second component contains 20,000 generated list-arithmetic examples adapted from the `num_list` style of NUMERICBENCH [LCX+25]. It allocates 4,000 examples each to minimum, maximum, sum, count, and sorting. Each prompt contains a short natural-language instruction over a list of two to five integers, with five paraphrase templates per operation; the completion is the scalar answer for minimum, maximum, sum, and count, or the sorted list for sorting. We use NU-

Table 7. Implementation choices for specialized number encoding.

Setting	FoNE	BitToken	BPE baseline
Numerical input	<num> token + value	<num> token + value	ordinary token IDs
Number representation	Fourier features, base 10	IEEE-754 float64 bits plus reciprocal	learned token embeddings only
Embedding integration	additive value embedding	zero-padded bit embedding	standard embedding lookup
Auxiliary numeric loss	digit cross-entropy	bitwise BCE	none
Negative numbers	not used in number-token training	used in number-token training	encoded as text tokens

MERICBENCH as the task-design citation but do not copy its records into TARITHTRAIN. Thus the rows used here are synthetic examples; the upstream benchmark is listed under the MIT license in Appendix C.

Free-form arithmetic. The third component contains 10,000 rows sampled from the GSM8K training split [CKB+21]. Prompts use the form `Question: ... Answer:` , and completions keep the original GSM8K rationale and final answer marker. The GSM8K dataset is released under the MIT license.

Natural-language anchor. The final component contains 10,000 examples from the TULU-3 SFT mixture [LMP+25]. These rows are included to reduce catastrophic drift on ordinary instruction-following language while the other 90% of the mix emphasizes arithmetic. For each selected conversation, we take the first user message as the prompt and the following assistant message as the completion. The collection is licensed under ODC-BY-1.0.

Length filtering. For the GSM8K and TULU-3 components, examples are kept only if the concatenated prompt and completion fit both tokenization regimes used in Section 6: at most 508 Olmo BPE tokens and at most 2,044 Bolmo byte tokens before the trainer adds its boundary tokens. This leaves four tokens of slack relative to the 512-token Olmo and 2,048-token Bolmo training limits and prevents train-time truncation.

F.2. Training Details

We run up to ten rounds of hyperparameter sweeping per model; the selected configurations are summarized below.

Shared training recipe. We use standard cross-entropy loss as the optimization objective of SFT. Both models are fine-tuned with the same trainer and optimization recipe; only the per-run values in Table 8 differ. We optimize with AdamW [LH19] (β_1=0.9, β_2=0.999, ϵ=10^{-8}, weight decay 0.01) under a cosine learning-rate decay with linear warmup over the first 5% of optimizer steps.

Gradients are accumulated to a global batch of 16. We train in `bfloat16`, pad each batch to a multiple of 64 tokens, and fix the random seed to 42.

Table 8. Bolmo-7B fine-tuning hyperparameters (run-specific values; all other settings follow the shared recipe).

Hyperparameter	Value
Base model	Bolmo-7B
Learning rate	1×10^{-6}
Max sequence length	2048 (byte tokens)
Warmup steps	5%
Max Gradient Norm	1

G. Additional Related Work

A smaller body of work aims to interpret how LLMs solve arithmetic tasks. Per-digit positional embeddings [MBS+24] and the choice of positional encoding [KPR+23] have a large effect on how well arithmetic extrapolates beyond the training length. [NRMB25], [BvGO25], [Cha24], and [ALP+25] reverse-engineer arithmetic in trained models and find heuristics, divisor lookups, one-digit lookahead, and sum-precision features in place of an algorithm. [DLS+23] show that transformers reduce compositional tasks to a flat subgraph match that breaks at moderate depth. [LJL+24] and [NCL+23] study when arithmetic appears during training and trace grokking to an implicit-bias shift and to a Fourier-plus-trigonometric algorithm. These works study positions, circuits, and training dynamics but treat the tokenizer as fixed. We do the opposite: hold the model fixed and vary the tokenizer, then report the natural-language cost.