Cross-Tokenizer LLM Distillation through a Byte-Level Interface

Avyav Kumar Singh[1*†], Yen-Chen Wu[1*], Alexandru Cioba[1‡], Alberto Bernacchia[1], Davide Buffelli[1],

[1]MediaTek Research, Cambridge (United Kingdom)

Correspondence: davide.buffelli@mtkresearch.com

Abstract

Cross-tokenizer distillation (CTD), the transfer of knowledge from a teacher to a student language model when the two use different tokenizers, remains a largely unsolved problem. Existing approaches rely on heuristic strategies to align mismatched vocabularies, introducing considerable complexity. In this paper, we propose a simple but effective baseline called **B**yte-**L**evel **D**istillation (BLD) which enables CTD by operating at a common interface across tokenizers: the byte level. In more detail, we convert the teacher's output distribution to byte-level probabilities, attach a lightweight byte-level decoder head to the student, and distill through this shared byte-level interface. Despite its simplicity, BLD performs competitively with—and on several benchmarks surpasses—significantly more sophisticated CTD methods, across a range of distillation tasks with models from 1B to 8B parameters. Our results suggest that the byte level is a natural common ground for cross-tokenizer knowledge transfer, while also highlighting that consistent improvements across all tasks and benchmarks remain elusive, underscoring that CTD is still an open problem.

1 Introduction

Large Language Models (LLMs) demonstrated unprecedented capabilities in natural language understanding, generation, and reasoning. Their applications are becoming ubiquitous, from conversational agents (e.g., (Guo et al., 2025; Yang et al., 2025; OpenAI, 2025)) and next-generation search engines (Xi et al., 2025) to tools that assist in scientific discovery (Zhang et al., 2024b) and software development (Dong et al., 2025). The remarkable performance of these models, however, is intrinsically linked to their scale with state-of-the-art LLMs often comprising billions of parameters. This size renders their training prohibitively expensive for most research institutes, and often inference becomes prohibitively slow for real-time or on-device applications.

To bridge the gap between the capabilities of large frontier models and the practical constraints of real-world systems, knowledge distillation has emerged as a seminal technique (Hinton et al., 2015). Distillation is a process in which a compact *student* model is trained to mimic the behavior of a larger, more powerful *teacher* model. Instead of learning solely from hard labels in a dataset, the student learns from the rich, dense output distribution produced by the teacher. This allows the student to inherit the teacher's sophisticated reasoning patterns while operating with a fraction of the computational footprint. The impact of distillation is already evident across the research environment and the industry, e.g., it enables to speedup the training of small specialized models, and to "compress" models and lower costs when serving them at scale (Xu et al., 2024).

Despite its success, the standard framework for knowledge distillation is built on a fundamental, yet restrictive, assumption: the teacher and student models must share an identical tokenizer and vocabulary. This is because the most common form of distillation operates at the *logit* level, where the student is trained to match the teacher's probability distribution over a fixed set of vocabulary tokens. If the tokenizers differ, their corresponding vocabularies lead to distinct output spaces. A logit vector of size 50,000 from the teacher cannot be directly compared to a logit vector of size 32,000 from the student. Consequently, performing *cross-tokenizer distillation* (CTD) has been considered infeasible without resorting to approximations or heuristics. These workarounds, such as distilling

[*]Equal contribution.

[†]Work done during an internship at MediaTek Research. Avyav is now at King's College London, London (United Kingdom).

[‡]Work done while at MediaTek Research. Alexandru is now at Orbital Materials, London (United Kingdom).

from generated text samples (Kim and Rush, 2016) or attempting to create ad-hoc mappings between vocabularies or hidden states (Boizard et al., 2025; Wan et al., 2024; Zhang et al., 2024a; Minixhofer et al., 2025), are either computationally inefficient, suffer from significant information loss, or lack a principled theoretical foundation.

The ability to perform principled CTD would unlock powerful new paradigms. First, it would allow us to combine the distinct strengths of diverse models. For instance, one could distill the broad world knowledge of a general-purpose model (e.g., trained with a large, multilingual tokenizer) into a specialized student model equipped with a domain-specific tokenizer optimized for medicine, law, or finance. This would create highly efficient and accurate expert models. Second, it would enable distillation from ensembles of heterogeneous models. For example, training a single student by distilling the collective intelligence of several top-tier open-source models (e.g., DeepSeek (Guo et al., 2025), Qwen (Yang et al., 2025), GPT-OSS (OpenAI, 2025), etc.), each with its own tokenizer. This would allow the student to learn a consensus of knowledge that potentially surpasses any individual teacher.

In this paper we introduce Byte-Level Distillation (BLD), which sidesteps the vocabulary mismatch in cross-tokenizer distillation by operating at the byte level—a representation shared by all tokenizers. Our method (i) converts the teacher's token-level output distribution to byte-level probabilities using a fast approximation (Vieira et al., 2025), (ii) attaches a lightweight, learnable byte-level decoder head to the student in parallel with its original token-level head, and (iii) performs distillation through this shared byte-level interface. After distillation, the byte-level head is simply removed, leaving a standard token-level model. This approach enables direct and effective knowledge transfer between models with different tokenizers.

Despite its simplicity, BLD performs competitively with—and on several benchmarks surpasses—substantially more complex CTD methods across tokenizer transfer and cross-model distillation tasks with models ranging from 1B to 8B parameters. At the same time, no method, including ours, achieves consistent gains across all benchmarks, suggesting that CTD remains an open and challenging problem. In summary, our contributions are:

- We propose BLD, a simple and alignment-free baseline for CTD that operates through a shared byte-level interface.
- We empirically show that this simple approach performs competitively with significantly more complex state-of-the-art CTD methods across a range of tasks.
- Through our analysis of the results, we highlight that no existing method—including ours—consistently dominates across benchmarks, and argue that CTD remains a largely open problem deserving further investigation.

2 Related Work

Our work is positioned at the intersection of three active areas of research: cross-tokenizer knowledge distillation, byte-level language modeling, and methods for converting token-level probability distributions to the byte level.

Cross-Tokenizer Distillation The challenge of transferring knowledge between models with different tokenizers is a significant hurdle for standard distillation techniques. Several recent works have proposed approximate or heuristic methods to bridge this gap. For instance, some approaches focus on aligning the vocabularies of the teacher and student models through various mapping strategies. Boizard et al. (2025) introduce a Universal Logit Distillation (ULD) loss based on optimal transport theory, which allows for distillation across different architectures and tokenizers without requiring them to share the same vocabulary. Other works, like Wan et al. (2024) and Zhang et al. (2024a), explore knowledge fusion and dual-space distillation, respectively, to enable knowledge transfer between heterogeneous models. Similarly, Minixhofer et al. (2025) propose a method for universal cross-tokenizer distillation through approximate likelihood matching. These methods often introduce additional complexity and rely on approximations to align the output spaces of the models. In contrast, our proposed BLD method circumvents this issue by operating at the byte level, a universal interface shared by all tokenizers.

Byte-Level Probability Estimation A core component of our BLD method is the ability to obtain a byte-level probability distribution from a standard token-based language model. This has been the focus of a number of recent studies. Vieira et al.

(2025) present algorithms for converting token-level language models into character-level ones. Phan et al. (2025) introduce the Byte-Token Representation Lemma, a framework that provides a formal mapping between a model's learned token distribution and its equivalent byte-level distribution. Our work leverages the insights from these works to create a shared byte-level space for distillation.

Byte-Level Language Models Our work is also related to the growing body of research on byte-level language models, which can be broadly categorized by how they process raw byte sequences. First are the pure byte-level models, which operate directly on sequences of bytes without any explicit grouping. Xue et al. (2022), with their ByT5 model, demonstrated that a standard Transformer architecture can be adapted to process byte sequences effectively, achieving competitive performance with token-level models while being more robust to noise. More recently, Wang et al. (2024) proposed MambaByte, a token-free model based on the selective state space architecture. Second are models that use fixed chunking to group bytes into patches. YU et al. (2023) introduced MEGABYTE, a multi-scale architecture that segments long byte sequences into fixed-size patches, using a local model within patches and a global model across them. Slagle (2024) proposed SpaceByte, which uses larger Transformer blocks after specific bytes (like spaces) to more efficiently model byte sequences. The autoregressive U-Net (AU-Net) of Videau et al. (2025) also falls into this category, as it pools bytes into a multi-scale representation based on fixed rules. Third are models that employ learned chunking to dynamically group bytes. Hierarchical Transformers like the Hourglass model from Nawrot et al. (2021) and the dynamic pooling mechanism from Nawrot et al. (2023) laid the groundwork for more flexible byte-level processing. More recent works have built on this, such as the Byte Latent Transformer (BLT) from Pagnoni et al. (2025), which encodes bytes into dynamically sized patches based on next-byte entropy, and MrT5 from Kallini et al. (2025), which uses dynamic token merging. The H-Net model from (Hwang et al., 2025) takes this a step further with a dynamic chunking mechanism that learns content- and context-dependent segmentation directly from the data, effectively creating an end-to-end, tokenizer-free model. While our method does not involve using byte-level models, it can be used to distill information from token based model into byte-level ones.

3 Our Method

3.1 Preliminaries

Let Σ be the alphabet containing all bytes, i.e., $\{1, 2, \ldots, 256\}$, and let Σ^* be the set of all sequences over the alphabet. Given a vocabulary $V \subseteq \Sigma^*$, which determines all the possible tokens, a tokenizer is a deterministic function that maps sequences of bytes to sequences of tokens: $\mathcal{T} : \Sigma^* \to V^*$, where V^* indicates the set of all sequences composed of tokens from the vocabulary V. We also define a decoder function $\mathcal{D} : V^* \to \Sigma^*$ as the function that "maps back" from a sequence of tokens to a sequence of bytes. We can assume that the decoder function is the inverse of the tokenizer, i.e., $\mathcal{D}(\mathcal{T}(\{b_1, b_2, \ldots, b_{N_b}\})) = \{b_1, b_2, \ldots, b_{N_b}\}$, with N_b indicating the length of the byte sequence, though this is not always the case in practice[1].

When performing distillation, the goal is to transfer knowledge from a teacher model to a student model. The teacher model has an associated vocabulary V_T, tokenizer $\mathcal{T}_T$, and decoder $\mathcal{D}_T$. The teacher model can be seen as a function mapping a given tokenized input sequence into a probability distribution over its vocabulary indicating the probability of the next token, $f_T : \mathcal{T}_T(\Sigma_T^*) \to \Delta(V_T)$, where $\Delta(V_T)$ is the probability simplex over the vocabulary. Similarly, the student model also has a vocabulary V_S, tokenizer $\mathcal{T}_S$, and decoder $\mathcal{D}_S$, which may differ from those of the teacher.

In standard distillation approaches, given a dataset of tokenized sequences $\mathcal{Z} = \{s_1, s_2, \ldots\}$, each one composed of multiple tokens $s_i = \{t_1, t_2, \ldots, t_{|s_i|}\}$, the student model parameters are updated by minimizing the following loss function

$$\mathcal{L} = \sum_{s_i \in \mathcal{Z}} \frac{1}{|s_i|} \Bigg(\sum_{t_j \in s_i} \mathrm{CE}(\delta(t_j), f_S(t_{<j})) + \mathrm{KL}(f_T(t_{<j}), f_S(t_{<j})) \Bigg) \quad (1)$$

where $\delta(t_j)$ is the delta function which is zero everywhere except at the index of token t_j for which it is equal to 1, $t_{<j}$ indicates the sequence

[1]This is because in practice tokenizers involve some pre-tokenization steps which are not reversible, like for example normalizing Unicode characters.

of tokens up to the j-th token excluded, CE indicates cross-entropy, and KL indicates the Kullback–Leibler divergence. The first term in equation 1, the cross entropy, is the standard next token prediction loss, while the second term, the KL divergence, is responsible for transferring knowledge from the teacher to the student. Notice however that for the latter to be well defined, it requires teacher and student to have the same vocabulary, which in practice usually leads to sharing also the same tokenizer, although in theory the it could be different between the two. Recently, several works have introduced heuristic or approximate strategies to overcome this issue (Boizard et al., 2025; Wan et al., 2024; Zhang et al., 2024a; Minixhofer et al., 2025). These approaches require identifying some form of alignment between tokenizations and introducing additional heuristic losses. Our approach instead overcomes these challenges by performing distillation at the byte level.

From BPE-level to Byte-Level Probabilities. Given a sequence of bytes $\{b_1, b_2, \dots, b_{N_b}\}$ and a teacher model f_T with vocabulary V_T and tokenizer $\mathcal{T}_T$, Phan et al. (2025) and Vieira et al. (2025) show that it is possible compute the probability of generating a sequence of bytes using the model f_T by summing the probabilities that the model assigns to all the *coverings* of the byte sequence. Let us define a *covering*, associated to the teacher model, for a byte sequence $\{b_1, b_2, \dots, b_{N_b}\}$ as the set containing all the sequences of tokens that "cover" the sequence of bytes when decoded, i.e.,

$$\begin{aligned}&\text{cover}_T(b = \{b_1, b_2, \dots, b_{N_b}\}) = \\ &\quad \{\{t_1, t_2, \dots, t_m\} \in V_T^* \quad | \quad \exists i \in \mathbb{Z}^{>0} \text{ s.t.} \\ &\quad \mathcal{D}_T(\{t_1, t_2, \dots, t_{m-1}\}) = b_{<i} \text{ and} \\ &\quad b_{\geq i} \text{ is a prefix of } \mathcal{D}(t_m)\}\}\end{aligned} \tag{2}$$

We can now compute the probability assigned by the teacher to a byte sequence $b = \{b_1, b_2, \dots, b_{N_b}\}$ as

$$P_T(b) = \sum_{y_i \in \text{cover}_T(b)} \prod_{t_j^{(i)} \in s_i} f_T\left(t_j^{(i)} | t_{<j}^{(i)}\right)$$

From this we can straightforwardly obtain the conditional probabilities for each single byte in the sequence as

$$P_T(b_i | b_{<i}) = \frac{P_T(\{b_1, b_2, \dots, b_i\})}{P_T(\{b_1, b_2, \dots, b_{i-1}\})} \tag{3}$$

The above procedure can be quite expensive computationally, but Vieira et al. (2025) provide a fast approximation, which we use for our method. More details are provided in Appendix C.

A naive approach to byte level CTD. Given that we can extract the probabilities at the byte level from any token based model, one might think of "going back" from byte level to a different token level to perform CTD. In fact, a naive approach for byte-level CTD, once the probabilities $P_T(b_i|b_{<i})$ at the byte level are extracted from the teacher for a given sequence, could be to use them to construct the probabilities of a tokenized version of the sequence in which the student's tokenizer is used instead. In more detail, given a sequence $b = \{b_1, b_2, \dots\}$, we can tokenize it using the student's tokenizer into a sequence of tokens $\{y_1, y_2, \dots\} = \mathcal{T}_S(b)$, and then compute the probability of each possible token (as this is needed for the KL term in the distillation loss) in V_S as follows

$$\begin{aligned}&\forall t = \{b_1^{(t)}, \dots, b_k^{(t)}\} \in V_S, \\ &\quad P(y_i = t | y_{<i}) = \prod_{b_j^{(t)} \in t} P_T\left(b_j^{(t)} | b_{<j}^{(t)}, y_{<i}\right)\end{aligned} \tag{4}$$

where, with a slight abuse of notation, we use $P_T\left(b_j^{(t)} | b_{<j}^{(t)}, y_{<i}\right)$ to indicate the probability assigned by the teacher to the j-th byte of token t given all previous bytes in the whole sequence. This quantity is computed using the equations presented above. The advantage of this approach is that there is no need to add any module to the original architecture of the student (which instead is required in our method). On the other side, this approach has several issues that make it impractical. First, equation (4) requires the computation of $|V_S|$ probabilities – which in practice is between 30000 and 250000 – for each token in the sequence (where the sequence is tokenized according to the student's tokenizer $\mathcal{T}_S$), which would be computationally prohibitive. Second, if the byte level probabilities are computed with an approximate method, the errors will compound when computing equation (4).

3.2 Byte-Level Interface for Distillation

Our method, called Byte Level Distillation (BLD), can be divided into two steps which we present below. A schematization of BLD can be found in Figure 1.

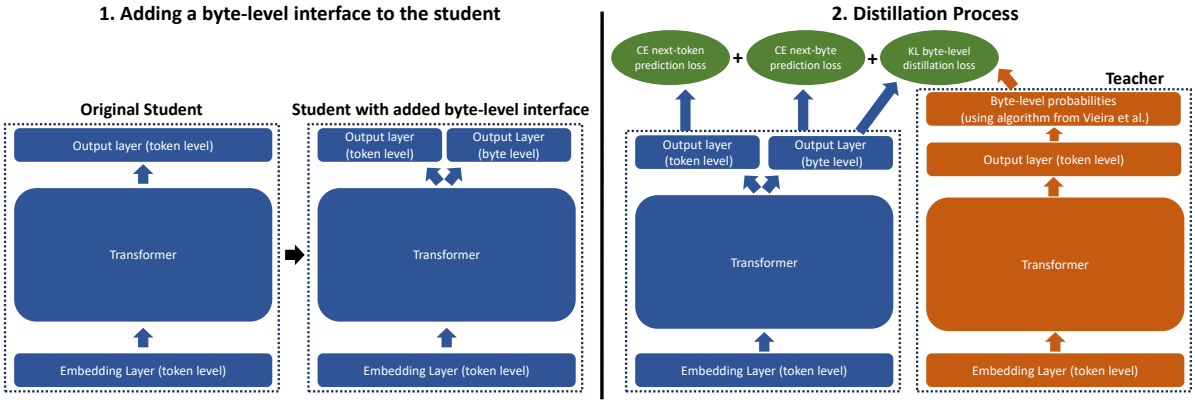

Figure 1: Representation of our Byte-Level Distillation (BLD) method composed of two steps. Step 1 adds a byte-level interface to the student model. Step 2 performs distillation by transferring knowledge from the teacher to the student using the shared byte-level interface. Additional next-token prediction and next-byte prediction losses are also used following standard distillation approaches. The byte-level interface can be removed at the end of the process.

Step 1: byte-level interface. The first step is to enable teacher and student models to share knowledge through the byte level. For the teacher we can use the approach presented in Section 3.1 to compute byte-level probabilities, but for enabling training of the student we need to introduce a new module to it. We start from a pretrained student model. The model is composed of a tokenizer $\mathcal{T}_S : \Sigma^* \to V_S^*$ with a respective decoder $\mathcal{D}_S : V_S^* \to \Sigma^*$, an encoder $E : V_S^* \to \mathbb{R}^{N\times d}$ (typically a learnable embedding matrix with one row for each element of the vocabulary V_S), a transformer $H : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$, and an output layer $O : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times |V_S|}$. Here N is the input sequence length (in terms of numbers of tokens from the vocabulary V_S), and d is the dimension of token embeddings and hidden representations (we assume they are the same for simplicity of presentation but in practice hidden dimensions at every layer of the transformer can be different from the dimensions of token embeddings). We now add a new learnable module to the student model. In more detail, in parallel to the existing token-level decoder O, we a add byte-level decoder: $O_b : \mathbb{R}^{N\times d} \to \mathbb{R}^{N_b\times|\Sigma|}$, where N_b is the length of the input sequence in terms of bytes. With this we have effectively added a *byte-level interface* to the output of the student model[2].

Step 2: distillation. Given a teacher model, we use the method of Vieira et al. (2025) to obtain $P_T(b_i|b_{<i})$ for each sequence $x_i = \{b_1, b_2, \dots\}$ in a given dataset $\mathcal{D}$. We can now perform distillation without requiring any specific alignment or heuristic as we have the probabilities at the byte-level obtained from the teacher model, and our student model has an output interface at the byte level. During distillation the loss is a combination of next-byte cross entropy loss, KL divergence at the byte level, and next-token cross entropy loss[3]. Formally, let $f_S : \mathcal{T}_S(V_S^*) \to \Delta(V_S)$ be the function at the token level for the student model obtained by composing $f_S(t) = O(H(E(t)))$ and let $f_S^{(b)} : \mathcal{T}_S(V_S^*) \times \mathbb{Z}^{>0} \to \Delta(\Sigma)$ be the function with the byte-level interface for the student model, i.e., $f_S^{(b)}(t, j) = O_b(H(E(t)))[j]$ (where "$[j]$" indicates selecting the j-th byte of the output), then the full loss for distillation is:

$$\mathcal{L} = \sum_{\substack{x_i\in\mathcal{Z},\\ \{t_1,t_2,\dots,t_k\}=\mathcal{T}_S(x_i),\\ t_i=\{b_1^{(i)},\dots,b_{n_i}^{(i)}\}}} \frac{1}{k}\sum_{\ell=1}^{k}\Bigg[\mathrm{CE}\left(\delta(t_\ell), f_s(t_{<\ell})\right) + \frac{1}{n_\ell}\sum_{j=1}^{n_\ell}\mathrm{CE}\left(\delta(b_j^{(\ell)}), f_S^{(b)}(t_{<\ell}, j)\right) + \mathrm{KL}\left(P_T(b_j^{(\ell)}|b_{<j}^{(\ell)}, t_{<\ell}), f_S^{(b)}(t_{<\ell}, j)\right)\Bigg] \quad (5)$$

where $P_T(b_j^{(\ell)}|b_{<j}^{(\ell)}, t_{<\ell})$ indicates the probability assigned by the teacher to the j-th byte in the ℓ-th token given all bytes in the sequence (including those from tokens prior to the ℓ-th) up to the $(j-1)$-

[2]The byte-level decoder can be pre-trained while keeping the rest of the weights fixed for additional stability, but in our experiments we found that it is not necessary.

[3]The next token cross entropy loss is added to ensure the weights of the token-level decoder O get updated.

th byte of the ℓ-th token. All or a subset of the parameters of the model can be updated during distillation, except from the byte-level output layer which must be updated if not pre-trained first.

After distillation, we remove the byte-level interface O_b, and thus keeping only the token-level output layer O. It is also possible to instead keep the byte level output layer if one is interested in generating outputs in terms of bytes or combinations of tokens and bytes. In our experiments, as byte-level decoder O_b, we use a simple linear projection for O_b with N_b fixed to 10, which means that for tokens that span more than 10 bytes, supervision signal will be provided only for the first ten. We validate our choice experimentally as shown in Appendix A. A different approach could be to have a small autoregressive layer to accomodate different values of N_b. We leave these directions for future work.

4 Experiments

To evaluate our approach, we follow the experimental procedure of Minixhofer et al. (2025) which considers three tasks: tokenizer transfer across different BPE tokenizers, tokenizer transfer from BPE to byte, and cross-tokenizer distillation.

Training setup. We fine-tune the student backbone with LoRA (Hu et al., 2022), applying rank $r = 64$ updates to the query and value projection matrices while keeping all other backbone weights frozen. For tokenizer transfer experiments, the embedding matrix and LM head are re-initialised using Fast Vocabulary Transfer (FVT) (Gee et al., 2022): tokens present in both vocabularies are initialised by directly copying the corresponding source embedding; tokens absent from the source vocabulary are initialised as the mean of their constituent sub-token embeddings, falling back to a random Gaussian sample drawn from the source embedding distribution when no decomposition is available. The byte-level decoder head O_b is a lightweight module consisting of 10 parallel linear projections from the model's hidden dimension to the byte vocabulary (260 tokens representing the 256 bytes and 4 special tokens for: beginning of sequence, end of sequence, padding, and out-of-vocabulary), enabling each token position to predict up to 10 bytes simultaneously (see Appendix A for a validation of this approach). We optimize with AdamW (Loshchilov and Hutter, 2019) using a cosine learning rate schedule with linear warm-up. Full hyperparameter details are provided in Appendix B. Importantly, we use the same SFT backbone for all considered distillation methods.

Training datasets For the BPE tokenizer transfer and byte tokenizer transfer experiments, we train on the Tulu-3 SFT mixture (Lambert et al., 2024). Byte-level teacher probabilities are pre-computed offline for this dataset using the fast approximation of Vieira et al. (2025), as described in Appendix C. For the cross-tokenizer distillation experiment (OpenMath2-Llama3.1-8B $\rightarrow$ Gemma2 2B), we train on the OpenMathInstruct-2 dataset (Toshniwal et al., 2024).

Validation datasets For the tokenizer transfer experiments, we use the no-robots split of Tulu-3 (Rajani et al., 2023) as a held-out validation set; this subset spans a diverse range of tasks—including coding, mathematics, and general reasoning—making it a representative signal for general-purpose capability. For the cross-tokenizer distillation experiment, we randomly sample approximately 1,000 examples from OpenMathInstruct-2 as a held-out validation set.

4.1 BPE Tokenizer Transfer

We first evaluate our method on the task of *tokenizer transfer* between two different BPE tokenizers. This involves selecting a pre-trained model, in our case LLama 3.2 3B (Meta, 2024) and replacing its tokenizer with the BPE tokenizer from Qwen 2 (Yang et al., 2024). The procedure involves replacing embedding and output projection layers with uninitialized layers in accordance (in terms of dimensionalities) with the new tokenizer, and distilling from the original model to the modified one. We present results in Table 1.

Table 1 shows that BLD performs competitively but does not uniformly dominate. It achieves the highest scores on PiQA (75.68) and AGI-ZH (35.97), and recovers performance close to the original model on PiQA, MMLU, and BoolQ, demonstrating that distillation through the byte-level interface successfully transfers general knowledge after tokenizer replacement. ALM + SFT is the strongest overall competitor, leading on four of seven benchmarks (ARC-C, BoolQ, MMLU, AGI-EN). The most notable weakness of BLD is instruction following: its IFEval score (30.58) lags far behind MinED (62.83) and ALM + SFT (58.51), both of which retain near-original IFEval performance.

Model	Method	Benchmark						
		PiQA	ARC-C	BoolQ	MMLU	AGI-EN	AGI-ZH	IFEval
original (Llama3.2 3B IT)		75.46	45.73	78.41	60.50	35.27	42.93	66.31
→ Qwen2	SFT	74.54	41.89	76.48	57.11	30.47	34.30	26.74
	DSKD	62.95	28.84	71.80	50.48	26.12	34.18	28.13
	MinED	75.35	42.58	78.65	58.20	34.68	34.76	**62.83**
	ALM + SFT	75.46	**45.82**	**79.36**	**58.86**	**36.64**	35.27	58.51
	BLCTD (Ours)	**75.68**	43.26	77.34	58.29	31.98	**35.97**	30.58

Table 1: Results of transferring Llama3.2 3B (Meta, 2024) to the Qwen2 tokenizer (Yang et al., 2024). *original* denotes the original model without transfer. *ARC-C* refers to Arc-Challenge. *AGI-EN* and *AGI-ZH* refer to the English and Chinese splits of AGIEval.

Model	Method	Benchmark						
		PiQA	ARC-C	BoolQ	MMLU	AGI-EN	AGI-ZH	IFEval
original (Llama3.2 3B IT)		75.46	45.73	78.41	60.50	35.27	42.93	66.31
→ Byte	SFT	67.30	31.57	**73.00**	38.95	26.05	35.18	24.70
	DSKD	64.47	31.31	60.34	37.62	23.74	33.36	23.98
	MinED	67.41	**32.94**	65.32	**39.84**	27.52	33.90	31.89
	ALM + SFT	66.32	31.57	71.41	39.15	**27.66**	**35.39**	29.74
	BLCTD (Ours)	**67.52**	30.89	69.85	39.06	26.44	34.57	25.43

Table 2: Results of transferring Llama3.2 3B (Meta, 2024) to byte-level tokenization. *original* denotes the original model without transfer. *ARC-C* refers to Arc-Challenge. *AGI-EN* and *AGI-ZH* refer to the English and Chinese splits of AGIEval.

This suggests that the byte-level distillation objective does not sufficiently preserve the structured output behaviour required for instruction following. DSKD performs worst across all benchmarks, confirming that direct distribution alignment without vocabulary alignment is ineffective in this setting.

4.2 BPE-to-byte Tokenizer Transfer

We now repeat the same *tokenizer transfer* task as the previous section, but this time we transfer from a BPE tokenizer to byte-level. This can be seen as adapting LLama 3.2 3B (Meta, 2024) to be a byte-level model. The procedure involves replacing embedding and output projection layers with uninitialized layers compatible with a byte-level tokenizer, and distilling from the original model to the modified one. We present results in Table 2.

Results in Table 2 show that transferring to byte-level tokenization is substantially harder than BPE-to-BPE transfer: all methods suffer large degradations across every benchmark (e.g., MMLU drops approximately 21 points and ARC-C approximately 13 points relative to the original model), reflecting the challenge of adapting a model trained on subword tokens to a much finer-grained representation. In this setting, BLD ranks first on PiQA (67.52), though the margin over MinED (67.41) is negligible. Performance leadership is fragmented across methods: SFT leads on BoolQ (73.00), MinED on ARC-C (32.94) and MMLU (39.84), and ALM + SFT on AGI-EN (27.66) and AGI-ZH (35.39). The spread between methods is noticeably narrower than in Table 1, suggesting that in this harder regime all approaches converge to a similar performance ceiling. DSKD again performs worst across most benchmarks. Overall, no method establishes a clear advantage, and the collective degradation relative to the original underscores that byte-level tokenizer transfer remains an unsolved challenge.

4.3 Cross-Tokenizer Distillation

Finally, we perform CTD across different models with different tokenizers. In more detail, we distill the maths-specialised OpenMath2-Llama3.1-8B (Toshniwal et al., 2024) into Gemma2 2B (Deepmind, 2024). Results are shown in Table 3.

Table 3 shows that BLD achieves the highest GSM8K score (62.55), modestly outperforming ALM + SFT (61.56) and SFT (59.29), and represents a meaningful gain over the uninitialised Gemma2 2B IT baseline (51.48). However, SFT leads on MATH (22.40 vs. 20.08 for BLD), suggesting that BLD's advantage over SFT is task-dependent and does not generalise uniformly across mathematical reasoning benchmarks. Despite BLD's result, the gap to the teacher (87.26 GSM8K, 37.60 MATH) remains very large, highlighting that effective cross-tokenizer knowledge transfer across heterogeneous models is still an open problem.

Model	Method	GSM8K	MATH
OpenMath2-Llama3.1-8B		87.26 $\pm$0.92	37.60 $\pm$2.16
Gemma2 2B IT		51.48 $\pm$1.38	10.60 $\pm$1.38
Gemma2 2B	SFT	59.29 $\pm$1.35	22.40 $\pm$1.87
	ALM + SFT	61.56 $\pm$1.34	19.00 $\pm$1.76
	Ours	**62.55** $\pm$1.33	20.08 $\pm$1.82

Table 3: Results of cross-tokenizer distilling the large math-specialized OpenMath2-Llama3.1-8B (Toshniwal et al., 2024) into the small Gemma2 2B (Deepmind, 2024) language model. All results are zero-shot CoT.

5 Limitations

Due to computational constraints, our work explores the task of tokenizer transfer with 3 billion parameter models, and the task of CTD between an 8 billion parameter teacher and a 2 billion parameter student. While these are practical sizes for models that are destined to run on-device, the behavior of CTD methods at larger scales remains underexplored.

Similarly, our distillation makes use of LORA to reduce the computational requirements, and performing full-parameter optimization may lead to higher performance.

6 Conclusions

In this paper we introduced BLD, a simple baseline for cross-tokenizer knowledge distillation that operates through a shared byte-level interface. By converting the teacher's output distribution to byte-level probabilities and attaching a lightweight byte-level decoder head to the student, our method avoids the complex vocabulary alignment procedures required by existing approaches. Despite this simplicity, BLD performs competitively with—and on several benchmarks outperforms—substantially more sophisticated methods across both tokenizer transfer and cross-model distillation settings. The effectiveness of this approach can be enhanced much further, for example, one can use a byte-level transformer architecture as opposed to MLP byte-level heads to capture sequential dependencies at the byte level.

Nevertheless, our experiments reveal a sobering finding: no method, including ours, achieves consistent improvements across all benchmarks and tasks. Performance leadership shifts depending on the benchmark, the transfer target, and the specific model pair. This inconsistency suggests that cross-tokenizer distillation remains a fundamentally open problem. We thus encourage the community to continue pursuing this line of research which has strong practical implications.

References

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

Google Deepmind. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *Preprint*, arXiv:2508.00083.

Leonidas Gee, Andrea Zugarini, Leonardo Rigutini, and Paolo Torroni. 2022. Fast vocabulary transfer for language model compression. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 409–416, Abu Dhabi, UAE. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*.

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

Julie Kallini, Shikhar Murty, Christopher D Manning, Christopher Potts, and Róbert Csordás. 2025. Mrt5: Dynamic token merging for efficient byte-level language models. In *The Thirteenth International Conference on Learning Representations*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 1317–1327.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengding Hu, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2024. Tülu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

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

Meta. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Benjamin Minixhofer, Ivan Vulić, and Edoardo Maria Ponti. 2025. Universal cross-tokenizer distillation via approximate likelihood matching. In *The Thirty-Ninth Conference on Neural Information Processing Systems*.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

Piotr Nawrot, Szymon Tworkowski, Michal Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2021. Hierarchical transformers are more efficient language models. *CoRR*, abs/2110.13711.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. 2025. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258, Vienna, Austria. Association for Computational Linguistics.

Buu Phan, Brandon Amos, Itai Gat, Marton Havasi, Matthew J. Muckley, and Karen Ullrich. 2025. Exact byte-level probabilities from tokenized language models for FIM-tasks and model ensembles. In *The Thirteenth International Conference on Learning Representations*.

Nazneen Rajani, Lewis Tunstall, Edward Beeching, Nathan Lambert, Alexander M. Rush, and Thomas Wolf. 2023. No robots.

Kevin Slagle. 2024. Spacebyte: Towards deleting tokenization from large language modeling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. 2024. Openmathinstruct-2: Accelerating AI for math with massive open-source instruction data. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Mathurin Videau, Badr Youbi Idrissi, Alessandro Leite, Marc Schoenauer, Olivier Teytaud, and David Lopez-Paz. 2025. From bytes to ideas: Language modeling with autoregressive u-nets. *Preprint*, arXiv:2506.14761.

Tim Vieira, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J. O'Donnell, and Ryan Cotterell. 2025. From language models over tokens to language models over characters. In *Forty-second International Conference on Machine Learning*.

Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. 2024. Knowledge fusion of large language models. In *The Twelfth International Conference on Learning Representations*.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. 2024. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*.

Yunjia Xi, Jianghao Lin, Yongzhao Xiao, Zheli Zhou, Rong Shan, Te Gao, Jiachen Zhu, Weiwen Liu, Yong Yu, and Weinan Zhang. 2025. A survey of llm-based deep search agents: Paradigm, optimization, evaluation, and challenges. *Preprint*, arXiv:2508.05668.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. A survey on knowledge distillation of large language models. *Preprint*, arXiv:2402.13116.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

LILI YU, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Songming Zhang, Xue Zhang, Zengkui Sun, Yufeng Chen, and Jinan Xu. 2024a. Dual-space knowledge distillation for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18164–18181.

Yu Zhang, Xiusi Chen, Bowen Jin, Sheng Wang, Shuiwang Ji, Wei Wang, and Jiawei Han. 2024b. A comprehensive survey of scientific large language models and their applications in scientific discovery. *Preprint*, arXiv:2406.10833.

A Evaluating the use of Linear Layers as Byte Level Heads

To test the effectiveness of a simple linear layer for each byte level head, we performed SFT *only at the byte level* on the Llama3.2 1B model (Meta, 2024) on a subset the TULU-3 dataset (Lambert et al., 2024). We then looked at training and validation losses over both bytes and tokens. We report the plots in Figure 2. We observe that, not only do the training and validation losses decrease smoothly for the byte level, but, surprisingly, they decrease also for the token level, demonstrating the effectiveness of adding even simple linear layers as heads for the byte level interface. This also indicates that a byte-level probability distribution can be effectively used for knowledge distillation – thus bridging a gap between different tokenizers with a common byte-level interface.

B Training Hyperparameters

We provide the values for the main hyperparameters used in our experiments, together with the respective search space for the tuning procedure in Table 4. The values for the baselines follow the optimized setup of Minixhofer et al. (2025), and only the learning rate has been further tuned due to computational constraints. For our method we tested different values of the weights for the loss functions.

C Approximation Settings for Byte-Probability Computations

The algorithm proposed by Vieira et al. (2025) provides an efficient approximation for computing byte-level probabilities from a token-level language model. In this section we describe the approximation parameters used in our implementation and the empirical procedure used to select them.

C.1 Approximation Parameters

The algorithm introduces two parameters, K and ϵ, that control the trade-off between computational efficiency and approximation accuracy when estimating the byte-level probability

$$P_T(b_1, b_2, \ldots, b_{N_b})$$

from a teacher model f_t operating over a token vocabulary (see Section 3.1).

Beam width (K). The algorithm performs a beam search over token sequences that are compatible with a given byte prefix. The beam width K specifies the maximum number of hypotheses retained during the search. Larger values of K allow more tokenization paths to be explored, which improves approximation accuracy but increases computational cost.

Pruning threshold (ϵ). During beam search, hypotheses with very small probability mass are removed. Specifically, beams whose probability falls below a threshold ϵ relative to the highest-probability beam are pruned. This pruning step eliminates tokenization paths that contribute negligibly to the final byte probability distribution.

Together, K and ϵ determine the number of tokenization paths considered during the computation.

C.2 Algorithm for Byte Probability Computation

The byte-level probability distribution at each position is computed using the following procedure:

1. **Initialization:** Create a beam state with parameters K (beam width) and ϵ (pruning threshold). The beam maintains a set of candidate tokenization paths, each with an associated probability weight.

2. **For each byte position i:**

   - **Compute distribution:** Call `logp_next()` to obtain the log probability distribution over the next 256 possible byte values. This operation marginalizes over all tokenization paths

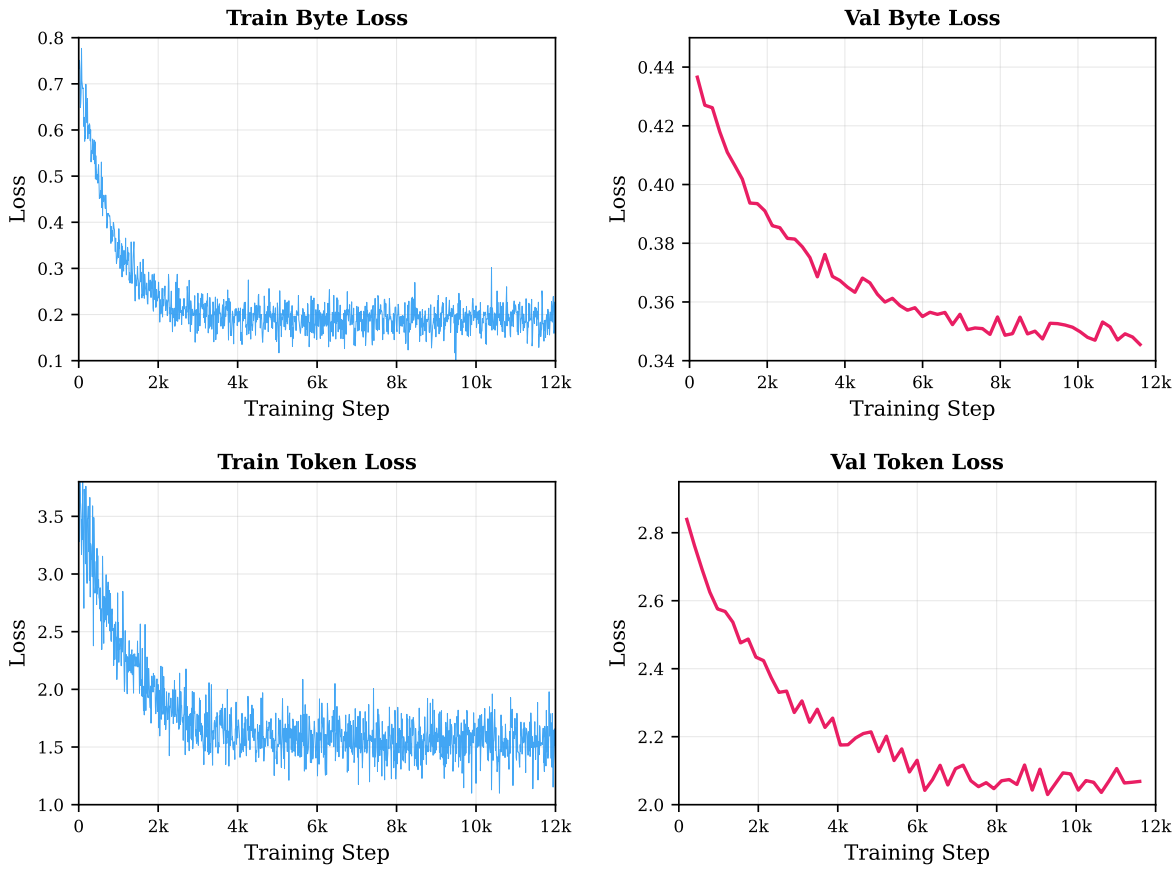

Figure 2: Training and validation losses for a Llama3.2-1B model with added byte-level head trained on a subset of the TULU-3 dataset with supervised fine-tuning only at the byte level. The top row plots are for training curves, while bottom row ones are for validation.

in the current beam:

$$\log P(b_i \mid b_{<i}) = \log \sum_{t \in \text{Beam}} P(t) \cdot P(b_i \mid t) \quad (6)$$

where t represents a tokenization path and $P(t)$ is its weight.

- **Advance beam:** Incorporate the observed byte b_i into the beam using the operation `beam.prune() « byte`. This extends each candidate path by consuming the byte.
- **Prune paths:** Remove tokenization paths with probability below the threshold ϵ relative to the highest-probability path. Retain at most K paths.
- **Handle token boundaries:** When a path completes a token, extend the beam by starting a new token using the teacher model's next-token probabilities.

The key computational bottleneck is the teacher model inference at token boundaries. The beam parameters K and ϵ control how many tokenization alternatives are maintained, which determines both accuracy and computational cost.

C.3 Evaluating Approximation Quality

We measure the Jensen–Shannon divergence (JSD) between the approximated byte probability distribution and a high-precision reference distribution computed using $K = 100, \epsilon = 10^{-6}$. Figure 3 shows the resulting approximation error for different combinations of K and ϵ. We observe that the setting $K = 10, \epsilon = 0.01$ achieves a Jensen–Shannon divergence of 0.0045. Figure 4 shows that runtime is primarily affected by the pruning threshold ϵ (lower values retain more beams), while beam width K has minimal impact due to efficient GPU batching of token queries. We also evaluate the effect of the approximation on downstream distillation performance by measuring the distilled model's perplexity and task accuracy, confirming that configurations with JSD < 0.005 produce negligible performance degradation.

Hyperparameter	Value	Search Space
LoRA		
Rank (r)	64	—
Alpha (α)	64	—
Dropout	0.05	—
Target modules	`q_proj`, `v_proj`, `k_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`	
Optimiser		
Algorithm	AdamW	—
Learning rate	2×10^{-5}	{5e-6, 1e-5, 2e-5, 3e-5, 5e-5, 1e-4}
Weight decay	0.01	—
(β_1, β_2)	$(0.9,\ 0.95)$	—
Gradient clipping (norm)	1.0	—
Learning rate schedule		
Scheduler	Cosine + linear warm-up	—
Warm-up steps	1,000	—
Training		
Epochs	5	—
Batch size (per device)	2	—
Gradient accumulation steps	4	—
Max sequence length	512	—
Precision	`bf16-mixed`	—
Loss coefficients		
KL divergence (λ_{KL})	0.1	$\{0.1,\ 0.2,\ 0.5,\ 0.8,\ 1.0\}$
Byte SFT (λ_b)	1.0	$\{0.5,\ 1.0\}$
Byte-level decoder head		
Parallel heads	10	—
Byte vocabulary size	261	—

Table 4: Training hyperparameters used in all experiments. The *Search Space* column lists the values explored during hyperparameter tuning; a dash indicates the value was fixed without search.

C.4 Experimental Setup

We conduct experiments using Llama-3.2-1B-Instruct and Llama-3.2-3B-Instruct as teacher models, with the Tulu-3 dataset for distillation. We test beam widths $K \in \{2, 5, 10, 20, 50, 100\}$ and pruning thresholds $\epsilon \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-6}\}$, measuring runtime and JSD relative to the reference configuration for each setting.

C.5 Parallel Implementation

Our implementation achieves efficient throughput through multi-level parallelization. The dataset is partitioned into shards, with each shard processed by an independent worker (one per GPU). Within each worker, we use a process pool with `n_sample_worker=15` to parallelize across samples, and the underlying trie operations batch up to 1000 token probability queries per forward pass. We use Python's `asyncio` framework to overlap CPU preprocessing with GPU computation.

In our experiments using four NVIDIA RTX 3090 GPUs, the configuration $K = 10, \epsilon = 0.01$ achieves approximately 10.4 seconds per sample for 100–150 byte sequences. We choose this configuration because it provides excellent approximation accuracy (JSD < 0.005) while using $10\times$ less memory than the reference configuration ($K = 100$), enabling higher sample-level parallelism. The lower memory footprint allows us to process more samples concurrently, and the balanced pruning threshold $\epsilon = 0.01$ avoids both

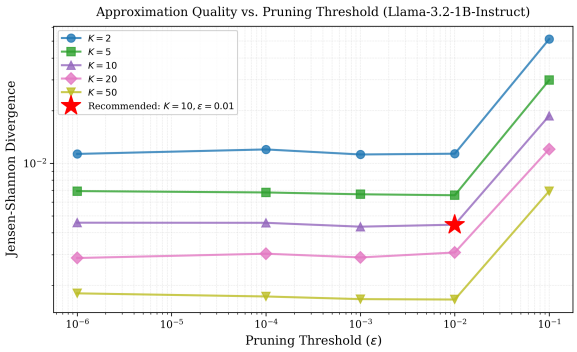

Figure 3: Jensen–Shannon divergence between approximated byte distributions and the reference distribution under different approximation settings.

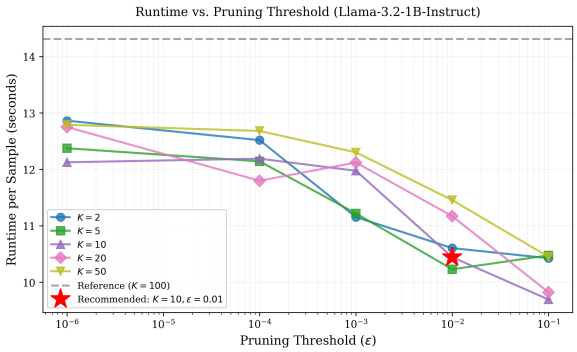

Figure 4: Runtime of byte-probability computation under different beam search configurations.

overly aggressive pruning (which degrades accuracy) and overly conservative retention (which increases memory usage). With this configuration and parallelism, computing byte probabilities for the entire Tulu-3 dataset requires approximately 2 days.