

Bolmo: Byteifying the Next Generation of Language Models

Benjamin Minixhofer^{♥1,2} Tyler Murray¹ Tomasz Limisiewicz³ Anna Korhonen² Luke Zettlemoyer³
 Noah A. Smith^{1,3} Edoardo M. Ponti^{♥4,2} Luca Soldaini^{♥1} Valentin Hofmann^{♥1,3}

¹Allen Institute for AI ²University of Cambridge ³University of Washington ⁴University of Edinburgh

♥ marks core contributors.

💡 **Models:** Bolmo-7B Bolmo-1B

💻 **Data:** Bolmo Mix

🔗 **Code:** bolmo-core

Abstract



We introduce **Bolmo**, the first family of competitive fully open byte-level language models (LMs) at the 1B and 7B parameter scales. In contrast to prior research on byte-level LMs, which focuses predominantly on training from scratch, we train Bolmo by *byteifying* existing subword-level LMs. Byteification enables overcoming the limitations of subword tokenization — such as insufficient character understanding and efficiency constraints due to the fixed subword vocabulary — while performing at the level of leading subword-level LMs. Bolmo is specifically designed for byteification: our architecture resolves a mismatch between the expressivity of prior byte-level architectures and subword-level LMs, which makes it possible to employ an effective exact distillation objective between Bolmo and the source subword model. This allows for converting a subword-level LM to a byte-level LM by investing less than 1% of a typical pretraining token budget. Bolmo substantially outperforms all prior byte-level LMs of comparable size, and outperforms the source subword-level LMs on character understanding and, in some cases, coding, while coming close to matching the original LMs' performance on other tasks. Furthermore, we show that Bolmo can achieve inference speeds competitive with subword-level LMs by training with higher token compression ratios, and can be cheaply and effectively post-trained by leveraging the existing ecosystem around the source subword-level LM. Our results finally make byte-level LMs a practical choice competitive with subword-level LMs across a wide set of use cases.

Contents

1	Introduction	3
2	Related Work	4
3	Byteified Olmo	5
3.1	Architecture	5
3.1.1	Non-Causal Patch Boundary Prediction	7
3.2	Byteifying Procedure	9
3.2.1	Stage 1: Subword-to-Byte Distillation	9
3.2.2	Stage 2: End-to-End Training	11
4	Experiment Setup	11
5	Main Results	12
5.1	Training at Higher Compression Factors	14
5.2	Post-Training Byteified Models via Task Arithmetic	16
6	Ablations	17
6.1	Impact of Non-Causal Patch Boundaries	17
6.2	Is Stage 1 Training Necessary?	17
6.3	Selecting the Right Local Model Architecture for Fast Inference	18
7	Conclusion	19
8	Future Directions	20
A	Additional Ablations	29
B	Benchmark Details	29
C	CUTE-Style Training Data	29
D	Does Post-Training Byteified Models via Task Arithmetic Always Work?	30
E	Embedding Rank Analysis	31
F	Full Hyperparameters	32

1 Introduction

Contemporary language models (LMs) predominantly rely on subword tokenization (Sennrich et al., 2016; Kudo, 2018) to segment text into a fixed vocabulary of tokens. This leads to many problems: they suffer from insufficient character understanding (Edman et al., 2024; Cosma et al., 2025; Uzan and Pinter, 2025), tokenization bias (Phan et al., 2024; Hayase et al., 2025; Vieira et al., 2025),¹ inability to incorporate sufficiently many words (e.g., across languages) in their fixed vocabulary (the *vocabulary bottleneck*; Liang et al., 2023), and potentially suboptimal compute allocation (Hwang et al., 2025; Pagnoni et al., 2025). These problems have motivated extensive research into alternatives to subword tokenization, most commonly by switching to UTF-8 bytes.² Many prior byte-level LMs claim to outperform subword-level LMs on some efficiency–performance Pareto frontier (Nawrot et al., 2023; Slagle, 2024; Wang et al., 2024; Hwang et al., 2025; Pagnoni et al., 2025; Zheng et al., 2025). However, in practice, byte-level LMs have not seen widespread adoption so far, with all leading LMs still exclusively relying on subword tokenization.

We hypothesize that the key reason for this mismatch between theory and practice is the fact that existing approaches to byte-level language modeling focus predominantly on training a new byte-level model from scratch, and compare against a compute-matched subword-level LM trained from scratch. In contrast, training of state-of-the-art subword-level LMs is rapidly evolving, combining innovations in training data curation, model architecture, and post-training strategies. Keeping this pace is unfeasible for byte-level LM development without extensive investments.

To resolve this mismatch, we introduce **Bolmo**, the first family of fully open byte-level LMs achieving performance on the level of state-of-the-art subword-level LMs across various tasks. In contrast to prior byte-level LMs which focus predominantly on training from scratch, Bolmo is trained by *byteifying* an existing subword-level LM using less than 1% of a typical pretraining budget (39.3B tokens). We train Bolmo 7B and Bolmo 1B by byteifying Olmo 3 7B (Olmo Team, 2025) and OLMo 2 1B (OLMo et al., 2024), respectively.

Bolmo follows the same overall architecture as the recent DTP (Nawrot et al., 2023), BLT (Pagnoni et al., 2025) and H-Net (Hwang et al., 2025) models, which we refer to collectively as Latent Tokenizer Language Models (LTLMs). However, we specifically adapt the Bolmo architecture to be well-suited to byteification (see Section 3.1). In particular, we resolve a mismatch between the expressivity of subword tokenization and the tokenization in LTLMs by allowing Bolmo’s latent tokenization to use future context (Section 3.1.1). Alongside an efficient two-stage training procedure starting with exact distillation of the source subword-level LM (Section 3.2), this allows for quickly recovering and in some cases surpassing the performance of the source subword-level LM. We believe that byteifying provides a key missing direction for research on byte-level LMs by enabling the creation of state-of-the-art byte-level LMs without extensive investments. Byteifying is complementary to training from scratch: making it cheap to byteify any subword model can quickly unveil high-performing architectures which are promising candidates for training from scratch as byte-level LMs.

Our Bolmo models on average outperform all prior public byte-level LMs of comparable size; for example, Bolmo 7B achieves +16.5% absolute improvement in STEM tasks over BLT 7B, which was trained from scratch. Bolmo 7B also greatly outperforms the source Olmo 3 on character understanding and, in some cases, on coding. In addition, Bolmo can be arbitrarily further sped up by training with higher compression ratios of bytes per patch, which is only possible to a limited extent in subword-level LMs (Section 5.1). Furthermore, we show that existing post-trained checkpoints can be utilized to post-train a byteified model without any additional training cost (Section 5.2); this allows for further speeding up research on byte-level LMs by re-using components in the source LM ecosystem. Finally, we provide extensive ablations on our design choices, analyzing the remaining gaps to subword-level LMs, as well as the gaps that Bolmo closed (Section 6).

Overall, byteifying lets us finally make byte-level LMs a practical choice competitive with subword-level LMs across a wide set of use cases. We hope the public Bolmo data, models, and code will help further advance research on byte-level language modeling.

¹Tokenization bias is the phenomenon that sequences of tokens can implicitly leak information about the future content of the text they tokenize. For example, assuming a vocabulary of English words, the token sequence `{_Hello, _Wor}` leaks that there is no `ld` after `_Wor`, otherwise the text would have been tokenized as `{_Hello, _World}` instead. This can lead to unintuitive behavior in practice (see Minixhofer et al., 2025b; Vieira et al., 2025).

²Graves (2013) may have been the first to model language over UTF-8 bytes; see Mielke et al. (2021) for an overview.

2 Related Work

Tokenization. LMs process information represented as a discrete sequence of symbols called *tokens* or *patches*. The process of segmenting the input into this discrete sequence is called *tokenization*, with different ways to tokenize being used across modalities such as text (Kudo, 2018), audio (Borsos et al., 2023) and images (Dosovitskiy, 2020). The predominant approach to tokenize text since the inception of LMs has been subword tokenization (Sennrich et al., 2016; Kudo, 2018): tokenizing text into a discrete sequence of units from a finite vocabulary of subword tokens (usually of size 30k-300k), typically represented as integer IDs. Subword tokenization causes a number of problems. **(i)** Information about the characters within each token is lost. While LMs have been shown to implicitly learn their tokens’ constituent characters (Kaushal and Mahowald, 2022; Edman et al., 2024) and it is possible to explicitly re-introduce character information (Cosma et al., 2025), they still fall short in tasks requiring character knowledge (Edman et al., 2024; Uzan and Pinter, 2025). **(ii)** The implicit reliance of subword tokenization on the future contents of the text (called *tokenization bias*) causes unexpected behavior at inference if the prompt ends in the middle of a word or with whitespace (Phan et al., 2024; Hayase et al., 2025; Vieira et al., 2025). **(iii)** The need for a fixed, finite subword vocabulary causes restrictive rigidity: for example, while encoding English efficiently is crucial for pretraining since the vast majority of current pretraining documents are in English, various downstream tasks have different efficiency requirements across different languages. **(iv)** Tokenization in contemporary LMs is tied to compute allocation: in a standard LM, the same amount of compute is spent on processing every token in the prefill, every token contributes equally to the KV cache size, and a fixed amount of compute is spent on sequentially generating any new token. Although there are ways to mitigate this problem post-hoc — such as KV cache sparsification (Łańcucki et al., 2025) and multi-token prediction (Gloeckle et al., 2024) — directly adapting the tokenization and thus the compute allocation based on the input instead might be more effective (Nawrot et al., 2023; Pagnoni et al., 2025).

Byte-level LMs. The shortcomings of subword tokenization have motivated extensive work on a wide range of alternatives, which even include tokenizing text by rendering it into pixels and segmenting these into patches (Lotz et al., 2023; Rust et al., 2023; Wei et al., 2025). The most common alternative has been tokenizing into a smaller set of finer-grained atomic units, such as UTF-8 bytes,³ instead. One strand of work directly replaces subword tokens with UTF-8 bytes, keeping other aspects of the architecture mostly the same (Xue et al., 2022; Wang et al., 2024; Minixhofer et al., 2025b; Zheng et al., 2025). This potentially solves problems **(i)** - **(iii)**⁴ of subword tokenization, but compute allocation remains a problem, exacerbated by having to process on average at least four times longer sequences of bytes. To mitigate this problem, some architectures pool a fixed amount of tokens into a single representation with a lightweight *local encoder* (e.g., another Transformer network), pass the pooled representations through a deep *global model* operating over the shortened sequence, then depool the representations back to the original granularity via a *local decoder*. This approach has been pioneered for autoregressive models by the Hourglass Transformer (Nawrot et al., 2022) and later adopted more broadly (Yu et al., 2023; Ho et al., 2024). Recent subsequent work has shown that replacing static pooling with dynamic tokenization improves the performance–efficiency Pareto front (Nawrot et al., 2023; Slagle, 2024). In this case, the token boundaries may be learned end-to-end, rely on entropy spikes, or be externally supervised (Nawrot et al., 2023; Hwang et al., 2025). We refer to these architectures as Latent Tokenizer Language Models (LTLMs) collectively, since — although operating over bytes — they perform a tokenization step inside the model which aggregates the byte representations into representations over latent patches. Byte-level LTLMs finally have the ability to address issues **(i)** - **(iv)** of subword tokenization. The most recent LTLMs have shown promise by performing on par with subword tokenization when spending the same total amount of FLOPs on training (Hwang et al., 2025; Pagnoni et al., 2025).

Tokenizer Transfer and Retrofitting. Techniques to alter a model’s architecture with extra training are typically referred to as *retrofitting*, which often relies on self-distillation (Bick et al., 2024; Łańcucki et al., 2025). The principal difficulty when this involves a change of tokenizer is finding embeddings for the new

³Although byte-level LMs are sometimes called ‘tokenizer-free’, it is more correct to say that UTF-8 is the tokenizer, and the vocabulary is the set of 256 distinct bytes.

⁴Since UTF-8 is designed primarily for the Latin script, problem (ii) of inefficiency in languages besides English might persist. However, alternative fine-grained units provide a promising alternative (Lemisiewicz et al., 2024; Land and Arnett, 2025).

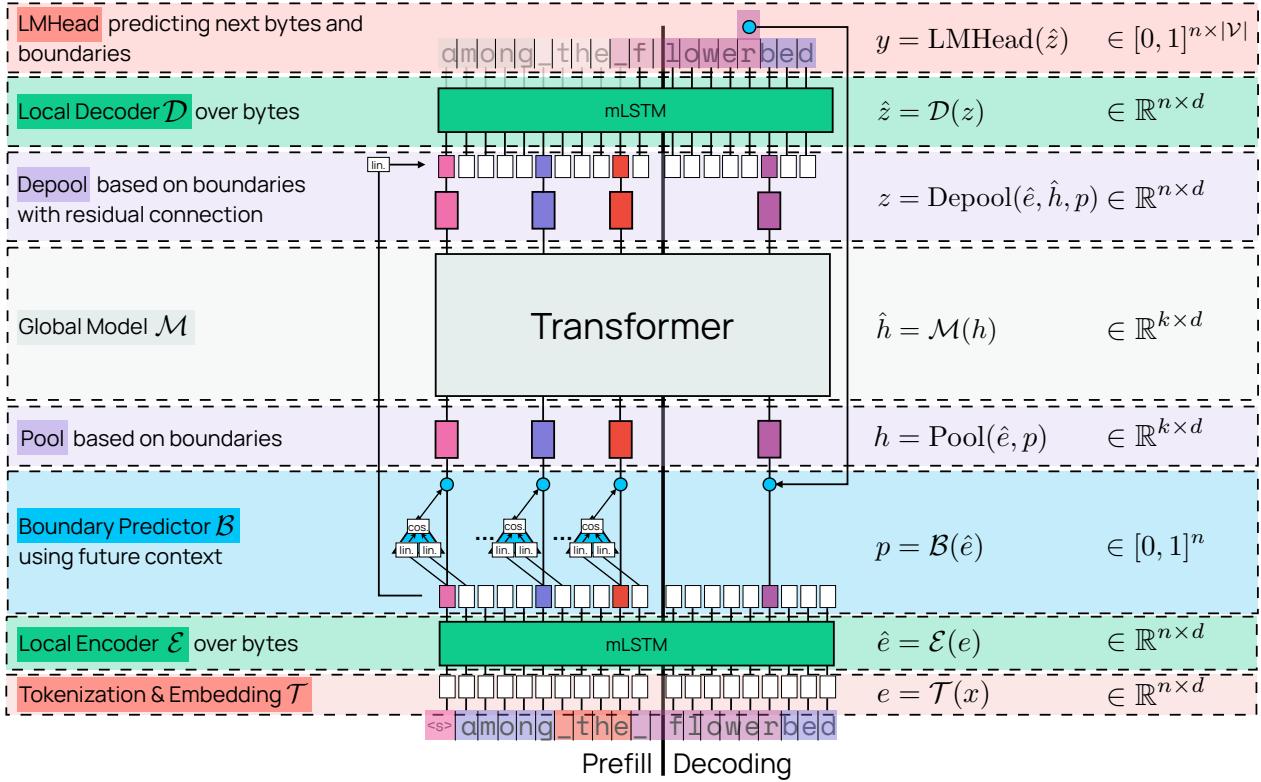


Figure 1 The Bolmo architecture. Tokenization & Embedding \mathcal{T} transforms the input text into one representation per byte. The representations are contextualized with the local encoder \mathcal{E} consisting of mLSTM blocks. The boundary predictor \mathcal{B} decides where to place patch boundaries using one byte of future context. The representations are then Pooled, passed through the global model \mathcal{M} consisting of Transformer layers, and Depooled. Finally, the local decoder \mathcal{D} consisting of another mLSTM stack contextualizes the depooled byte representations and the LMHead transforms them into next-byte predictions, alongside deciding where to place the next patch boundary.

tokens; this is usually done using heuristics (Tran, 2020; Minixhofer et al., 2022; Dobler and de Melo, 2023) or training-based methods (Minixhofer et al., 2025a). Recently, effective tokenizer transfer methods based on *cross-tokenizer distillation* have been introduced (Dobler et al., 2025; Haltiuk and Smywiński-Pohl, 2025; Minixhofer et al., 2025b). Here, the original model is seen as the teacher, the tokenizer-transferred model is seen as the student, and the objective is to match the behavior of the student to the teacher. Byteification is a special case of tokenizer transfer. Byteification was first done by Pagnoni et al. (2025) by initializing the LTLM parameters from an existing subword model where possible and training as if from scratch. Hwang et al. (2025) later byteified by supervising the boundary prediction to match the subword boundaries and introducing an auxiliary embedding-matching loss. Our key contribution is creating an LTLM which is specifically suited to byteifying. We do so by introducing a novel architecture (Section 3.1), as well as a dedicated two-stage procedure to byteify efficiently by first learning to exactly recover the behavior of the source subword model (Section 3.2). Together, these innovations first allow closely matching the performance of state-of-the-art subword-level LMs with a byteified model.

3 Bytefied Olmo

3.1 Architecture

Following the same overall structure as prior LTLMs, Bolmo can be formalized as shown in Figure 1.

Tokenization & Embedding. \mathcal{T} assigns every input UTF-8 byte in x ⁵ a corresponding embedding in \mathbb{R}^d from an embedding table containing an entry for every byte. The embedding table over bytes is negligible in size compared to embedding tables over subwords. However, scaling the size and sparsity of the embedding table has been shown to improve performance while having no negative effect on inference speed (Huang et al., 2025). Inspired by BLT’s hash embeddings (Tito Svenstrup et al., 2017; Pagnoni et al., 2025), we thus increase the size of the embedding table. Specifically, we residually add the longest subword embedding (of the original subword-level LM’s embedding table) which ends at the current byte position to every byte embedding:

$$e_i := \mathcal{T}_{\text{Byte}}(x_i) + \mathcal{T}_{\text{SubwordSuffix}}(x_{:i})$$

where $\mathcal{T}_{\text{SubwordSuffix}}$ assigns an embedding to every byte based on the index of the subword token in the vocabulary $\mathcal{V}_{\text{Subword}}$ with the longest common suffix to the byte sequence up to the current position i . Retaining the subword embeddings is not strictly necessary, and we can generally achieve the same performance by increasing the size of the local encoder instead. However, subword embedding retention allows us to achieve a better performance–efficiency tradeoff by increasing the amount of cheap sparsely activated parameters.⁶

Local Encoder. The local encoder \mathcal{E} contextualizes the byte-level embeddings through an mLSTM layer (Beck et al., 2025a), resulting in the contextualized representations \hat{e} . We find that mLSTM improves inference speed compared to other linear RNN variants (see Section 6.3) while attaining competitive performance. We found a single mLSTM layer to be sufficient since the expressivity of the local encoder is substantially enhanced by the retained subword embeddings.

Boundary Predictor. The boundary predictor \mathcal{B} predicts a score $p \in [0, 1]$ for every byte based on the contextualized representations \hat{e} . If p is greater than some threshold, a patch boundary is placed after the current byte. In contrast to prior LTLMs, Bolmo’s boundary predictor is *non-causal*:⁷ it has access to one byte of future context, and it is only employed for the prefill, where future information can be used while retaining the ability to generate text. We describe non-causal boundary prediction in detail in Section 3.1.1, where we also discuss how boundary prediction is handled during decoding.

Pooling. We pool byte-level representations into patch representations by selecting the representation of the last byte in every patch as the patch-level representation h . This is equivalent to the pooling done by Hwang et al. (2025),⁸ and does not introduce any extra parameters. Contrary to Hwang et al. (2025), the local models and the global model use the same representation dimensionality, obviating the need for an upprojection.⁹

Global Model. The majority of compute is spent in the deep global model \mathcal{M} contextualizing the patch representations h into \hat{h} . We retain the global model of the original subword-level LM, i.e. the Olmo 3 decoder-only transformer backbone.

Depooling. The global model is invoked at every patch boundary, providing a contextualized representations for every patch. It remains to depool these representations back to representations of bytes. We do so by adding the latest available patch representation in \hat{h} at any byte position to a linear projection of the byte representations \hat{e} , resulting in z . This is similar to Hwang et al. (2025)’s depooling, again forgoing the projection due to equal global and local dimensionality.

⁵We treat x as a sequence over bytes, i.e. $x \in \{0, \dots, 255\}^n$.

⁶An alternative to increasing the size and sparsity of the local encoder is using a mixture of experts in the feed-forward layer, although we do not investigate this here.

⁷For consistency with prior work, we use the term ‘non-causal’ to contrast with ‘causal’ as in causal language models, i.e., causal in the sense of using only unidirectional context, although this is arguably a misnomer.

⁸Hwang et al. (2025) refer to the process of creating a single representation for every patch as *routing*, whereas we refer to this more generally as *pooling*, which also encompasses the cross-attention pooling done by Pagnoni et al. (2025).

⁹We originally experimented with smaller local dimensions but found the upprojection mechanism to bottleneck performance by restricting the rank of the representations (see Appendix E).

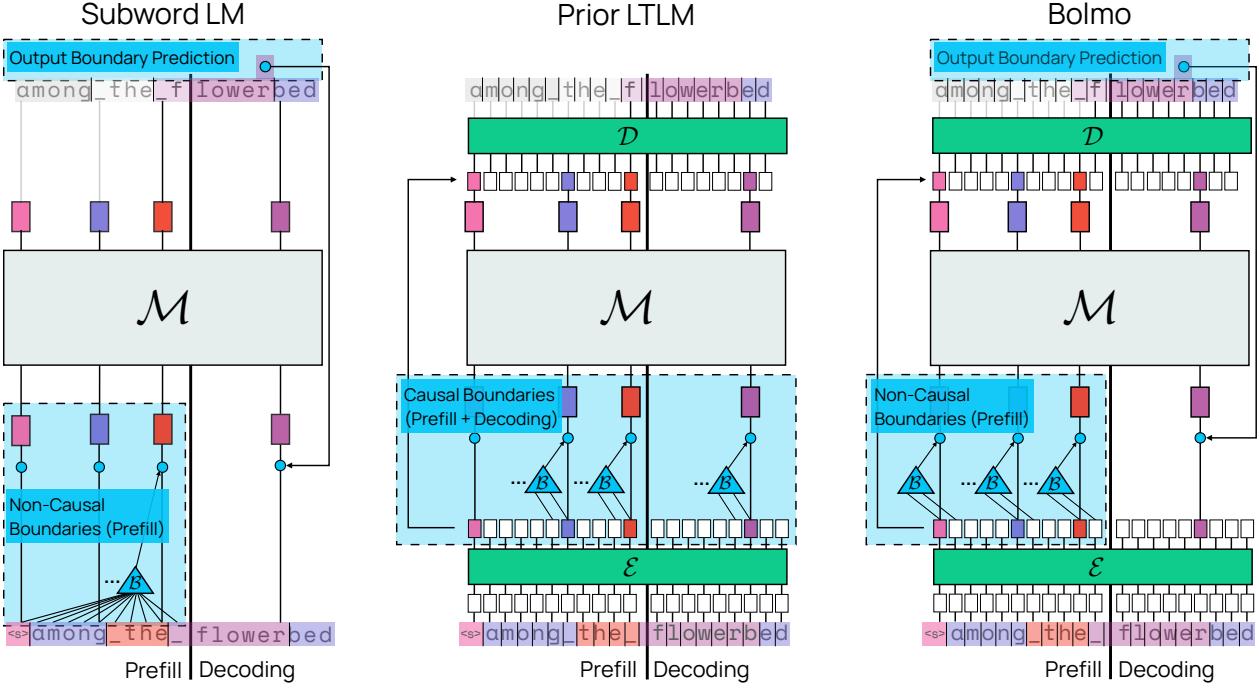


Figure 2 Subword-level LMs non-causally set boundaries over the prefill using the external subword tokenizer, then implicitly predict boundaries alongside the text content during decoding (*left*). Prior byte-level LTLMs causally set boundaries with a light-weight boundary predictor during both prefill and decoding (*middle*). We restore the expressivity of subword-level LM boundaries by non-causally predicting boundaries for the prefill, then predicting whether a boundary occurs alongside the next byte during decoding (*right*).

Local Decoder. The local decoder \mathcal{D} contextualizes the depooled byte representations z into \hat{z} via another stack of mLSTM layers. Here, we use a larger number of mLSTM layers (in practice, four) to increase capacity since unlike in the encoder, we find it infeasible to meaningfully re-incorporate the output subword embedding matrix, which could have potentially allowed reducing the number of layers in the decoder in a similar way as for the encoder.

Language Modeling Head. The language modeling head LMHead converts the final byte representations \hat{z} into scores interpretable as next-byte probabilities via a projection to the vocabulary space and softmax.

Overall, our modifications keep the total parameter count similar to the parameter count of the source subword-level LM by removing the output embedding matrix but adding new parameters from the local encoder layers and local decoder layers. In practice, Bolmo 1B contains ~ 10 M fewer parameters than OLMo 2 1B (-0.7%), and Bolmo 7B contains ~ 330 M more parameters than Olmo 3 7B ($+4.5\%$).

3.1.1 Non-Causal Patch Boundary Prediction

Prior LTLMs employ a causality constraint on the boundary predictions: the boundary predictor only uses past context to decide on whether to place a boundary.¹⁰ At a glance, this seems necessary: we are aiming to predict the next byte, so we must not leak any information about it. However, although subword-level LMs employ a causality constraint over the subword tokens, the subword tokens themselves do not depend exclusively on past context: *subword tokenizers use information about future bytes to place token boundaries*. To see this, let us interpret our subword tokenizer as a function which decides whether to place a token boundary after any byte, i.e. $\mathcal{B}(x) : \{0, 1, \dots, 255\}^n \rightarrow \{0, 1\}^n$. Let us assume a vocabulary of English words and subwords, the example text `_Hello_Wor!`, which would typically be tokenized as `{_Hello, _Wor, !}`, and the position $i = |\text{_Hello_Wor!}| - 1 = 9$. $\mathcal{B}(\text{_Hello_Wor!})_i = 1$ since there is a boundary after `r`. However, in

¹⁰The causality constraint on boundaries is referred to as *incrementality* by Pagnoni et al. (2025).

the text `_Hello_World!`, which would be tokenized as $\{\text{_Hello, _World, !}\}$, we have $\mathcal{B}(\text{_Hello_World})_i = 0$, despite $\text{_Hello_Wor!}[:i] = \text{_Hello_World!}[:i] = \text{_Hello_Wor}$. In other words, although the subword-level LM only uses past subword tokens to predict the next subword token, the subword tokens themselves are created by taking future context into account. In this case, this means deciding that `_Wor` should be a token in one case but not in the other, although the text up until that point is equivalent. Current LTLMs, in contrast, can not take future context into account. This creates a mismatch between the expressivity of LTLM boundary predictors and subword tokenizers. We modify the boundary predictor to resolve this mismatch. In particular, while prior boundary predictors are implemented as

$$\mathcal{B}(\hat{e})_t := f(\hat{e}_0, \hat{e}_1, \dots, \hat{e}_t)$$

we implement our boundary predictor as

$$\mathcal{B}_{\text{Bolmo}}(\hat{e})_t := f(\hat{e}_0, \hat{e}_1, \dots, \hat{e}_t, \hat{e}_{t+1}).$$

That is, we use up to `one byte of future context`. Concretely, we parametrize our boundary predictor as

$$\mathcal{B}_{\text{Bolmo}}(\hat{e})_t := \frac{1}{2} \left(1 - \frac{(W_q \hat{e}_{t+1})^T (W_k \hat{e}_t)}{\|W_q \hat{e}_{t+1}\| \|W_k \hat{e}_t\|} \right) \in [0, 1],$$

i.e., we compute the cosine distance between a projection of the representation of the current byte and the byte one position in the future. An equivalent parametrization, although using the current byte and one byte before, is used by Hwang et al. (2025). Taking one future byte into account largely resolves the mismatch between subword tokenizers and LTLM tokenization.¹¹ As shown in Figure 2, taking future context into account can also make patches more semantically coherent: for example, in the case of texts containing compounds such as `the flowerbed`, a boundary predictor has three intuitive options: (i) make the entire compound a single patch `flowerbed`, (ii) place a patch boundary after `r` to create the patch `flower`, or (iii) place a patch boundary after `b` (once it is evident this is a compound word) to create `flowerb`. Option (ii) is arguably the semantically most coherent one,¹² however, for a causal boundary predictor, this would mean having to place a patch boundary after `r` for every text starting with `the flower`, including e.g. `the flowers`, while a non-causal one can adjust based on future context. The byteification strategy of Hwang et al. (2025) supervises based on option (iii), i.e. predicting the start of the next subword token instead of the end of the previous one, which would create a patch `flowerb` as shown in Figure 2.

Output Boundary Prediction. While using future context is fine for prefilling, we need to know whether to place a boundary without observing the next byte for decoding. We thus add a special symbol `` to the vocabulary and let the local decoder learn to emit `` at the end of every patch (the local encoder, in contrast, never sees ``).¹³ In effect, we end up with two boundary predictors: the boundary predictor \mathcal{B} ingesting the shallowly contextualized representations from the local encoder with future context (used during prefill), and a boundary predictor as part of the language modeling head ingesting deeply contextualized representations from the local decoder without future context (used during decoding). Notably, this is precisely equivalent to what happens in subword-level LMs: the prefill is tokenized using the external subword tokenizer (analogous to the boundary predictor \mathcal{B}), and output boundaries `` are implicitly predicted alongside the text contents of every subword token upon decoding (analogous to our output boundary predictor) as illustrated in Figure 2.

¹¹ Subword tokenizers in principle have unrestricted access to the future, while we use a single byte. In practice, we find one byte of lookahead largely sufficient to match the behavior of subword tokenization. However, we believe future work on larger (or unrestricted) lookaheads could be fruitful.

¹² It is not clear whether human notions such as semantic coherence or faithfulness to linguistics should play a role in designing language models, see e.g. Beinborn and Pinter (2023); Minixhofer et al. (2023).

¹³ It is worth noting that predicting the boundary symbol `` is analogous to the output boundary prediction in Fleshman and Durme (2023), although the motivation differs.

Boundary Symbol Fusion. Since we are aiming to predict the symbol `` after every patch, our local decoder is turned from an isotropic model to a transducer from $\mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{(n+k) \times d}$, i.e., the local decoder needs to process k more positions. Although this overhead is not prohibitive in principle, it makes it difficult to compare models which use output boundary prediction and models which do not. We thus make it effectively zero-cost by doubling our byte vocabulary size from 256 to 512, for every byte adding a version of the same byte followed by a boundary. The goal of the local decoder, then, is to predict the current byte *and* whether it is followed by a boundary at every step. Fusing the boundary symbol turns the local decoder back into an isotropic model. The only remaining overhead is that the softmax has to be applied over a set of 512 instead of 256 output tokens, which is negligible.

On end-to-end learning of non-causal boundaries. Hwang et al. (2025) train the boundary predictor end-to-end by incorporating it in the computation graph through (i) smoothing of the contextualized global representations \hat{h} using the boundary scores and (ii) a straight-through estimator of the boundary scores applied to the depooled representations z . Training the boundary predictor end-to-end in this style is not immediately possible using our non-causal formulation. This is the case since (i) our output boundary predictor would need to estimate the precise boundary score assigned by the boundary predictor for decoding, instead of only predicting whether a boundary occurs or not and (ii) relatedly, instead of the single bit of information leaked by discrete boundary predictions, the model can learn to leak 16 bits of information (assuming we use bfloat16) about the next byte, which is enough to uniquely identify it. This could cause the model to learn degenerate solutions by exploiting the boundary scores to pass information about the future to the local decoder. In this work, we thus focus exclusively on strategies to train the boundary predictor with external supervision instead, which we believe have been underutilized in prior work.

3.2 Byteifying Procedure

We byteify by initializing the parameters of the global model from the subword-level LM checkpoint, while parameters of the local models and the LM head are initialized randomly. Our byteifying procedure consists of two stages. In the *first stage*, we aim to quickly learn weights for the local encoder, local decoder, boundary predictor and LM head which exactly recover the behavior of the subword-level LM. The parameters of the global model stay frozen in this stage. In the *second stage*, we train the entire model to let it learn to utilize byte-level information, while also optionally increasing the target compression ratio of bytes per patch.

3.2.1 Stage 1: Subword-to-Byte Distillation

The aim of the first stage is quickly learning weights for the local encoder, local decoder, boundary predictor and LM head which recover the behavior of the subword model. Efficiency is crucial; the cost of this stage should be minimal to permit fast experimentation and allow increasing the investment into Stage 2. To achieve these goals, we design a Stage 1 procedure which allows learning the desired weights without fully backpropagating through the global model. This substantially reduces the time per training step (see Appendix F). The Stage 1 loss is minimal if and only if the byte-level model exactly mimics the source subword-level LM. It is composed of three parts.

Quickly Learning a Boundary Predictor $\mathcal{B}_{\text{Bolmo}}$. We train the boundary predictor to emulate the boundaries placed by subword tokenization via a binary cross-entropy loss, i.e.,

$$\mathcal{L}_{\mathcal{B}} := - \sum_t (\mathcal{B}_{\text{subword}}(x)_t \log \mathcal{B}_{\text{Bolmo}}(\hat{e})_t + (1 - \mathcal{B}_{\text{subword}}(x)_t) \log(1 - \mathcal{B}_{\text{Bolmo}}(\hat{e})_t)),$$

where $\mathcal{B}_{\text{subword}}(x)$ is 1 for every byte at the last position of a subword patch, otherwise 0. The boundary predictor $\mathcal{B}_{\text{Bolmo}}$ utilizing future context to tokenize the prefill text quickly achieves >99% accuracy.

Quickly Learning a Local Encoder \mathcal{E} . Assuming our boundary predictor perfectly emulates subword tokenization, our local encoder and pooling mechanism will be a perfect substitute for the subword embedding matrix if they yield the same input to the global model as the subword embedding matrix for every patch. This is

the case if all pooled representations $\text{Pool}(\hat{e}, \mathcal{B}_{\text{Bolmo}}(\hat{e}))$ are equal to the corresponding subword embeddings $\mathcal{T}_{\text{Subword}}(x)$. Hwang et al. (2025) optimize toward this goal by directly minimizing the L2 distance of every pooled representation to the corresponding subword embedding. We take an alternative approach inspired by research on model stitching which shows that similar representations do not necessarily propagate through subsequent layers in a similar way (Athanasiadis et al., 2025). We propagate the pooled representations through n layers of the global model and minimize L2 distance to the subword representations which result from propagating the subword embeddings through the same n layers,

$$\mathcal{L}_{\mathcal{E}} := \|\mathcal{M}_{:n}(\text{Pool}(\mathcal{E}(\hat{e}, \mathcal{B}_{\text{subword}}(x))) - \mathcal{M}_{:n}(\mathcal{T}_{\text{subword}}(x))\|.$$

Notably, we pool the local encoder representations using the true subword boundaries $\mathcal{B}_{\text{subword}}$ instead of $\mathcal{B}_{\text{Bolmo}}$.¹⁴ $\mathcal{M}_{:n}$ indicates the global model up to and including the n -th layer. The weights of \mathcal{M} are kept frozen. If $n = 0$, this reduces to the setting of Hwang et al. (2025). Although choosing $n > 0$ necessitates backpropagating through some parts of the global model, we can minimize the resulting cost by choosing a small n . We find $n = 4$ to strike a good balance between performance and efficiency, substantially outperforming $n = 0$ while remaining cheap to compute.

Quickly Learning a Local Decoder \mathcal{D} . Our local decoder and LM head are optimal if our byte-level LM assigns the same likelihood as the subword model to every text x . Assuming equal patch boundaries, it is optimal if the likelihood of every *patch* is equal. Since subword-level LMs implicitly predict output patch boundaries, we cannot easily compute comparable patch likelihoods in byte-level models without output boundary prediction. In this case, we would have to resort to approximations as in Minixhofer et al. (2025b). However, since Bolmo does predict output patch boundaries, simply comparing the likelihoods of every patch results in an exact objective (i.e., a loss which is minimal if and only if both models are the same),

$$\mathcal{L}_{\mathcal{D}, \text{Distill}} := \sum_i f \left(\prod_{j \in T(x, i)} \text{LMHead}(\hat{z}_{\text{subword}})[j, \text{next_byte}(x, j)], \text{LMHead}_{\text{subword}}(z_{\text{subword}})[i, \text{next_tok}(x, i)] \right),$$

where $j \in T(x, i)$ indicates all byte indices j which are part of the i -th subword patch; this includes the indices of the special `` symbol if treated as separate, or the indices of the 256 special symbols consisting of a byte plus `` if fused. `next_tok(..)` and `next_byte(..)` map to the index in the vocabulary of the symbol occurring after the current symbol (token or byte), including special symbols.¹⁵ $z_{\text{subword}} = \mathcal{M}(\mathcal{T}_{\text{subword}}(x))$ are the representations of the subword model at the final layer, $\hat{z}_{\text{subword}} = \mathcal{D}(\text{Depool}(\hat{e}, z_{\text{subword}}, p))$ is the result of passing these representations through the depooling layer and the local decoder, and $\text{LMHead}_{\text{subword}}$ is the LM head of the source subword-level LM. As the comparison function f , we choose the temperature-modulated binary cross-entropy,

$$f(\hat{y} \parallel y) := - \left(y^{1/\tau} \log \hat{y}^{1/\tau} + (1 - y^{1/\tau}) \log (1 - \hat{y}^{1/\tau}) \right),$$

with $\tau = 5$ as recommended by Minixhofer et al. (2025b). In practice, we conduct the operations involved in the computation of $\mathcal{L}_{\mathcal{D}}$ in log-space to ensure stable numerics. We optionally combine the distillation loss $\mathcal{L}_{\mathcal{D}, \text{Distill}}$ with a cross-entropy loss to encourage modeling the training data well and to already start exploiting byte-level information,

$$\mathcal{L}_{\mathcal{D}, \text{CE}} := \sum_j -\log \text{LMHead}(\hat{z}_{\text{subword}})[j, \text{next_byte}(x, j)].$$

¹⁴Using the true subword boundaries instead of the boundaries predicted by $\mathcal{B}_{\text{Bolmo}}$ is necessary to preserve the alignment of the pooled representations to the representations in $\mathcal{T}_{\text{subword}}(x)$ along the sequence dimension.

¹⁵For example, $-\log \text{LMHead}_{\text{subword}}(..(x))[i, \text{next_tok}(x, i)]$ is the cross-entropy of the subword model.

Putting It Together. In principle, the boundary predictor and local encoder on the one hand, and the local decoder and LM head on the other, could be trained separately (assuming we stop the gradient to the encoder through \hat{z}_{subword}). Although there may be scenarios where this is beneficial, we choose to train them together for simplicity. The complete Stage 1 loss is given by

$$\mathcal{L}_{\text{Stage1}} := \lambda_{\mathcal{B}} \mathcal{L}_{\mathcal{B}} + \lambda_{\mathcal{E}} \mathcal{L}_{\mathcal{E}} + \lambda_{\mathcal{D}, \text{Distill}} \mathcal{L}_{\mathcal{D}, \text{Distill}} + \lambda_{\mathcal{D}, \text{CE}} \mathcal{L}_{\mathcal{D}, \text{CE}},$$

where $\lambda_{\mathcal{B}}, \lambda_{\mathcal{E}}, \lambda_{\mathcal{D}, \text{Distill}}, \lambda_{\mathcal{D}, \text{CE}} \in \mathbb{R}$ are the loss weights which we set $\lambda_{\mathcal{B}} = 4, \lambda_{\mathcal{E}} = 1, \lambda_{\mathcal{D}, \text{Distill}} = 1, \lambda_{\mathcal{D}, \text{CE}} = 1$. Stage 1 needs in total one forward pass through all layers and one backward pass through the first n layers of the global model, plus forward and backward passes through local encoder, local decoder, boundary predictor and LM head. This makes Stage 1 substantially more efficient than training the entire model. It could also be further optimized by quantizing or applying inference-specific optimizations to the global model layers starting from the $(n+1)$ -th layer (which we do not need to backpropagate through). We analyze the difference between inserting Stage 1 and directly training the entire model end-to-end with randomly initialized parameters (besides the global model) later in Section 6.2. Besides performance improvements, Stage 1 provides a vehicle for rapid experimentation: We can conduct Stage 1 training to rapidly check whether a particular architecture for the local encoder and decoder has sufficient capacity to emulate the input and output embedding matrices, respectively. We use this to guide the architecture search for Bolmo under the hypothesis that byte-level architectures which can not emulate the subword model after Stage 1 will remain inadequate with further Stage 2 training.

3.2.2 Stage 2: End-to-End Training

In the second stage, we train the entire model end-to-end, retaining only the boundary loss $\mathcal{L}_{\mathcal{B}}$ and the cross-entropy loss $\mathcal{L}_{\mathcal{D}, \text{CE}}$. For $\mathcal{L}_{\mathcal{D}, \text{CE}}$, we substitute the depooled representations \hat{z}_{subword} of the subword model representations with the true depooled representations \hat{z} , referring to this loss as \mathcal{L}_{CE} ,

$$\mathcal{L}_{\text{Stage2}} := \lambda_{\mathcal{B}} \mathcal{L}_{\mathcal{B}} + \lambda_{\text{CE}} \mathcal{L}_{\text{CE}}.$$

We now optimize all parameters, including those of the global model \mathcal{M} . This stage is intended for the model to adjust to the end-to-end setting, since in Stage 1 we assumed a local encoder and boundary predictor perfectly emulating the subword model, which, although close, is not true in practice. The global model can learn to exploit the new byte-level information in Stage 2, and optionally be trained with higher compression ratios of bytes per patch (see Section 5.1).

4 Experiment Setup

Data. The Bolmo data mix consists of $\sim 172\text{B}$ tokens¹⁶ from the Dolma 3 pretraining data mix (Olmo Team, 2025), augmented with 75M tokens of CUTE-style data (Edman et al., 2024), sampled so as not to overlap with the CUTE test set, to encourage character understanding (see Appendix C for details). Training runs for less than one epoch on this mix.

Model. We use the pretrained Olmo 3 7B checkpoint after mid-training and long-context extension (Olmo Team, 2025) as our starting point for byteifying into Bolmo. For the local models, we use stacks of alternating mLSTM (Beck et al., 2025a) and feedforward layers of size 1 and 4 for the encoder and decoder, respectively. See Appendix F for details on the architecture.

Training. For *Stage 1*, we train on a total of 9.8B tokens ($\approx 43\text{B}$ bytes). In this stage, we train the local encoder, decoder, boundary predictor and LM head, keeping the global model frozen. For *Stage 2*, we train the entire model on a total of 39.3B tokens ($\approx 173\text{B}$ bytes). See Appendix F for detailed training hyperparameters.

¹⁶We count tokens as tokenized by the Dolma2 Tokenizer.

Baseline. We compare against the Olmo 3 7B checkpoint with continued training on the Bolmo training data such that the amount of total gradient updates to the global model parameters is the same (i.e., on 39.3B tokens) to disentangle the effects of continued training with the same architecture and byteification.

Ablations and Development. We developed Bolmo primarily through experiments on OLMo 2 (OLMo et al., 2025). We optimized decisions around the architecture through quick Stage 1 training runs on OLMo 2 1B or 7B. Our byteifying procedure was then applied without adjustments to Olmo 3 7B. Since there is currently no 1B version of Olmo 3, we conduct experiments requiring larger sweeps across training configurations on OLMo 2 1B.

Evaluation. We create the Bolmo 7B evaluation suite based on Olmo Team (2025)’s OLMOBASEEVAL, skipping GSM Symbolic and BigCodeBench due to their size, and adding CUTE (Edman et al., 2024) and EXECUTE (Edman et al., 2025) to measure character understanding in English and across other languages, respectively. We create the Bolmo 1B evaluation suite based on Olmo Team (2025)’s Base Easy Suite, again adding CUTE (Edman et al., 2024) to measure character understanding. For the Bolmo 1B suite, we define a set of core tasks consisting of ARC (Clark et al., 2018), MMLU (Hendrycks et al., 2021), CSQA (Talmor et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), SocialIQA (Sap et al., 2019), PIQA (Bisk et al., 2020), the Basic Skills benchmark (Olmo Team, 2025) and CUTE (Edman et al., 2024) for use in ablations and sweeps (see Appendix B for details).

5 Main Results

Bolmo 7B Results. Table 1 compares Bolmo 7B with existing byte-level LMs of comparable size: EvaByte 6.5B (Zheng et al., 2025), TFree-Hat 7B (Neitemeier et al., 2025) and BLT 7B (Pagnoni et al., 2025), as well as the source Olmo 3 model (Olmo Team, 2025). Bolmo 7B performs best among all publicly known byte-level models in every category, including code, math, multiple-choice QA, and character understanding. As the only exception, Bolmo 7B slightly trails TFree-Hat 7B in the GenQA category (70.9 vs. 71.3). Bolmo 7B also comes close to matching the performance of the source Olmo 3 model (which is itself competitive with other subword-level LMs of comparable size; see Olmo Team, 2025). The remaining gap to Olmo 3 is largely not specific to byteifying; it can be attributed to continued training in general, see Appendix A.

On code, Bolmo 7B outperforms Olmo 3 due to higher pass@16 rates at generally slightly lower pass@1. This indicates that Bolmo 7B generates more diverse continuations than Olmo 3 under the given sampling settings, which are equivalent for both models (temperature = 0.6, top_p=0.6, see Appendix B). However, although promising, at this point we cannot conclude that byte-level models are fundamentally better suited to generating more diverse continuations, since we have not comprehensively explored the quality–diversity tradeoff at different points defined by different sampling strategies.

The character understanding results are surprising, as Bolmo’s accuracy vastly surpasses its subword-level counterpart. In fact, prior byte-level models do not outperform the subword Olmo 3 model. This could be explained by the hypothesis that character understanding is primarily acquired through scale (in terms of parameters and training tokens; Cosma et al., 2025), so although byte-level models should require less scale to acquire character understanding, the increased scale of Olmo 3—likely trained on substantially more tokens than the other models—might compensate for this.¹⁷ In contrast, Bolmo 7B is trained with synthetic data encouraging character understanding (Appendix C), which speeds up the acquisition of this skill. Bolmo 7B still outperforms Olmo 3 in a comparison where Olmo 3 had continued training on the Bolmo data mix for the same total amount of tokens (Appendix A), further suggesting that while character understanding is driven by scale, it emerges sooner in byte-level models.

Bolmo 1B Results. Table 2 compares Bolmo 1B (trained off of OLMo2 1B; OLMo et al., 2024) with existing byte-level models, including H-Net (for which no 7B checkpoint is available; Hwang et al., 2025) and BLT 1B. Although trained on the previous Olmo generation, Bolmo 1B is competitive with prior byte-level models of similar size, outperforming H-Net and slightly trailing behind BLT 1B, although BLT 1B has substantially

¹⁷ Not all training token/byte counts of prior byte-level models are public.

	Fully-Open Byte-Level LMs	Open-Weight Byte-Level LMs			Subword LMs
		Bolmo 7B	EvaByte 6.5B	TFree-Hat 7B	
# Parameters (incl. Embed)	7.63B	6.49B	7.19B	10.55B	7.30B
Char	75.1	47.3	47.9	49.3	<u>56.0</u>
CUTE	78.6	50.8	54.2	52.3	<u>56.9</u>
EXECUTE	71.6	43.8	41.6	46.3	<u>55.1</u>
Code	40.7	31.2	36.9	31.6	<u>39.5</u>
HumanEval pass@1/@16	40.6 / 74.7	34.7 / 49.1	41.1 / 61.4	31.5 / 44.7	49.0 / <u>71.1</u>
DeepSeek LeetCode pass@1/@16	2.3 / 7.6	1.6 / 3.3	0.9 / 4.6	1.2 / 4.8	<u>1.6</u> / <u>6.2</u>
DS 1000 pass@1	14.9	7.1	<u>18.2</u>	17.0	20.1
MBPP pass@1/@16	42.8 / 68.0	42.9 / <u>59.2</u>	44.6 / <u>59.2</u>	37.2 / 53.2	<u>44.3</u> / 54.9
MultiPL HumanEval pass@1/@16	26.8 / 62.5	16.8 / 31.8	<u>26.9</u> / 49.7	24.1 / 43.7	33.6 / <u>56.3</u>
MultiPL MBPP pass@1/@16	38.0 / 69.2	36.5 / <u>60.5</u>	38.8 / 60.3	36.0 / 54.6	37.8 / 59.9
Math	48.9	27.0	35.8	15.7	55.3
GSM8K	68.0	36.7	60.7	24.2	73.1
MATH	29.8	17.3	10.9	7.3	37.5
MC STEM	65.5	54.0	62.1	49.0	66.3
ARC MC	88.5	74.2	90.0	65.5	<u>89.2</u>
MMLU STEM	57.0	44.3	55.2	41.6	59.5
MedMCQA MC	47.8	37.9	51.5	37.6	<u>48.2</u>
MedQA MC	42.4	27.2	20.5	22.5	<u>42.0</u>
SciQ MC	91.9	86.6	93.3	77.8	<u>92.8</u>
MC Non-STEM	75.8	63.8	66.5	56.6	77.7
MMLU Humanities	67.2	52.7	57.4	52.2	69.2
MMLU Social Sci.	74.0	57.4	72.0	54.0	75.2
MMLU Other	65.1	50.9	65.0	49.3	66.9
CSQA MC	73.6	91.4	<u>75.4</u>	52.2	75.2
PiQA MC	79.4	65.3	<u>79.8</u>	62.9	80.3
SocialIQA MC	79.1	<u>79.6</u>	79.3	50.6	80.4
CoQA Gen2MC MC	90.0	63.2	<u>91.2</u>	71.4	92.9
DROP Gen2MC MC	59.1	41.1	24.9	31.0	62.5
Jeopardy Gen2MC MC	84.8	67.8	90.5	77.5	<u>85.5</u>
NaturalQs Gen2MC MC	65.9	43.8	70.7	50.1	<u>69.6</u>
SQuAD Gen2MC MC	95.8	88.8	25.8	71.0	96.8
GenQA	70.9	41.4	71.3	68.4	72.4
HellaSwag RC	78.8	70.1	82.8	<u>81.1</u>	77.8
Winogrande RC	85.5	78.2	88.2	88.2	<u>85.7</u>
Lambada	71.1	62.9	70.5	72.8	68.0
Basic Skills	89.6	82.7	<u>89.6</u>	84.5	90.0
DROP	65.2	7.8	48.6	38.8	71.5
Jeopardy	56.8	13.1	68.3	<u>67.3</u>	60.3
NaturalQs	28.6	5.4	34.3	29.2	32.6
SQuAD	91.6	35.9	88.6	85.2	93.5
CoQA	70.5	16.7	<u>71.1</u>	68.6	72.7

Table 1 Results comparing Bolmo 7B to existing byte-level models of comparable size and the source subword model (Olmo 3 7B) on the Bolmo 7B evaluation suite. All models except Bolmo were trained from scratch. **Boldface** indicates the best result per task, underline the second best.

more than one billion parameters since Pagnoni et al. (2025) do not count the hash embedding parameters. Like Bolmo 7B, Bolmo 1B exhibits performance degradation compared to the source subword model on some tasks, e.g. -3.2% on MMLU. However, on other tasks, Bolmo 1B outperforms OLMo2 1B, e.g. +5.1% on Lambada, +3.3% on CoQA and +32.5% on CUTE.

	Fully-Open Byte-Level LMs	Open-Weight Byte-Level LMs			Subword LMs
	Bolmo 1B	H-Net XL (1-stage)	H-Net XL (2-stage)	BLT 1B	OLMo 2 1B
# Parameters (incl. Embed)	1.47B	1.27B	1.60B	4.53B	1.48B
Bolmo 1B Suite	58.2	52.5	53.2	58.5	<u>58.3</u>
ARC	59.0	<u>61.8</u>	62.3	59.9	61.4
MMLU	37.2	37.5	38.7	40.6	40.4
CSQA	64.2	61.4	62.4	69.2	66.0
HellaSwag	67.0	60.2	63.6	71.0	68.9
WinoGrande	<u>65.7</u>	58.9	60.9	67.0	65.2
SocialIQA	<u>54.7</u>	50.1	52.9	54.6	55.1
PiQA	74.9	73.6	74.0	77.3	<u>76.4</u>
CoQA	81.7	73.7	72.8	81.7	77.4
DROP	<u>43.1</u>	33.4	33.6	37.7	52.7
Jeopardy	69.6	72.3	70.8	79.5	<u>76.4</u>
NaturalQs	40.9	34.5	35.9	<u>46.6</u>	46.8
SQuAD	<u>83.4</u>	76.7	77.1	76.4	87.4
SciQ	85.0	85.8	88.3	87.0	<u>87.5</u>
QASPER	<u>63.0</u>	<u>63.0</u>	51.1	59.6	64.0
Basic Skills	<u>73.3</u>	55.9	58.5	78.0	72.9
DBQA	26.5	27.7	<u>27.5</u>	<u>27.5</u>	25.2
ProtocolQA	<u>27.8</u>	28.7	25.9	28.7	<u>27.8</u>
Lambada	<u>65.2</u>	48.1	49.4	65.9	60.1
MedMCQA	30.5	30.9	<u>32.1</u>	33.1	31.1
MedQA	26.1	29.3	<u>28.2</u>	26.2	28.0
SciRIFF	<u>82.5</u>	73.8	80.5	81.1	85.0
CUTE	60.0	17.4	24.2	<u>37.6</u>	27.5

Table 2 Results comparing Bolmo 1B to existing byte-level models of comparable size and the source subword model (OLMo2 1B) on the Bolmo 1B evaluation suite. All models except Bolmo were trained from scratch. **Boldface** indicates the best result per task, underline the second best.

5.1 Training at Higher Compression Factors

Takeaway. Byteified models can be sped up by adapting the external boundary supervision to encourage a higher number of bytes per patch during training. This creates a way to smoothly trade off efficiency and performance which does not exist for subword-level LMs due to the softmax bottleneck.

A substantial advantage of LTLMs is — unlike subword-level LMs — not to be restricted to a fixed, finite set of patches. So far, we have not exploited this advantage since our primary goal was mimicking the tokenization of the source subword model. We now investigate whether we can leverage the increased freedom in our choice of the patching strategy to train a faster model by encouraging a higher average number of bytes per patch. In particular, we experiment with ways to change the external boundary supervision from the original subword tokenization boundaries $\mathcal{B}_{\text{subword}}$ to a subset of those boundaries. We fix a compression ratio t of target average bytes-per-patch. We then remove subword boundaries (i.e., merge subword tokens) of $\mathcal{B}_{\text{subword}}$ until the desired compression ratio is achieved. We experiment with three merging strategies.

- **BPE.** We iteratively merge the most common pair of tokens as in Byte Pair Encoding (Sennrich et al., 2016). In contrast to conventional BPE, we apply BPE per-example instead of over the entire corpus.¹⁸ This is inspired by the work of Feher et al. (2025), which has shown that it is possible to retrofit language models to operate over BPE merges of the tokens in their vocabulary.
- **Entropy.** We use a small auxiliary 370M parameter subword-level LM¹⁹ to compute next-token entropies for

¹⁸Although applying BPE per minibatch would also be possible we choose to apply it per-example to avoid nontrivial dependencies on the batch size.

¹⁹The auxiliary 370M parameter subword-level LM was trained on 74.3B tokens following a downscaled version of the OLMo 2 training and architecture (OLMo et al., 2024).

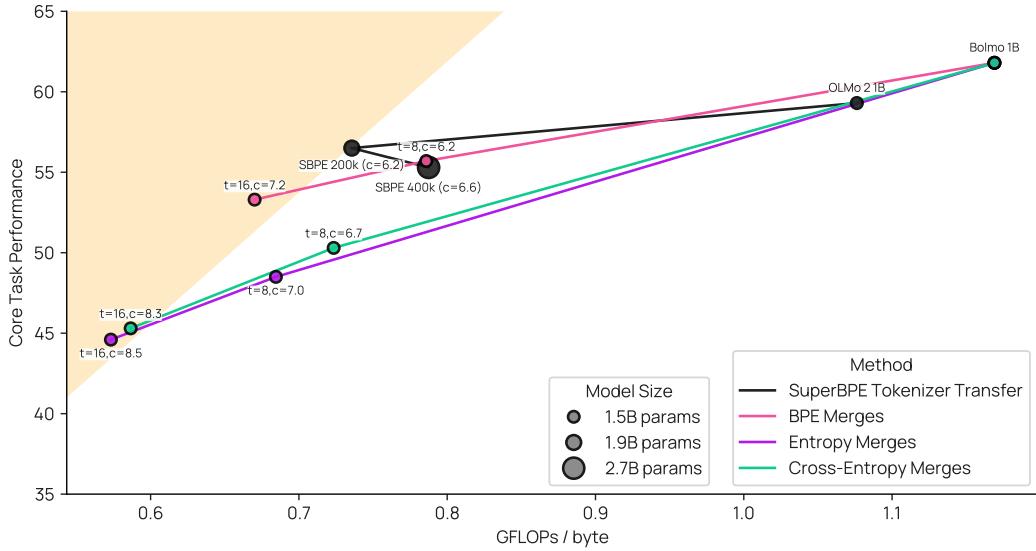


Figure 3 The task performance vs. efficiency Pareto frontier of (i) the source subword-level LM with tokenizer transfer to SuperBPE to achieve higher compression in bytes per patch and (ii) Bolmo models with adapted boundary prediction to achieve higher compression (see Section 5.1). The subword-level LM breaks off the frontier as the cost of the softmax starts to dominate for larger vocabulary sizes; byte-level LMs take over the frontier at that point, as seen in the optimal region around the top-left corner.

every token. We then iteratively merge the pair of patches which, when summing their individual entropies, results in the lowest entropy among all entropy sums of pairs of patches in the example.

- **Cross-Entropy.** We use the same small auxiliary LM as for entropy-based merging, but instead of merging the pair of tokens with the lowest total entropy, we iteratively merge the pair of tokens with the lowest total *cross-entropy* w.r.t. the next token in the data.

In the case of entropy- and cross-entropy-based merging, the auxiliary LM is only required at training time to supervise the boundary predictor (as in DTP; Nawrot et al., 2023). Unlike BLT (Pagnoni et al., 2025), we do not need to retain the auxiliary LM for inference.

Even though the loss is discontinuous w.r.t. the parameters of the boundary predictor and we do not employ any technique to backpropagate through the discrete boundary predictions, we observe stable training without loss spikes with all of the above merging methods. An important nuance is that the supervision target compression ratio t is not attained by the model. Despite the boundaries not being learned end-to-end, the model learns to trade off boundary prediction accuracy with the main next-byte prediction loss, like other multitask models which learn to balance performance on the constituent tasks (see e.g. Zhang and Yang, 2021). An important hyperparameter is thus the factor λ_B controlling the importance of the boundary prediction task; we keep $\lambda_B = 4$ from Stage 1 training and report the attained compression ratio c in addition to the target compression ratio t .

As the baseline, we increase the bytes per patch of the subword-level LM via tokenizer transfer to SuperBPE tokenizers (Liu et al., 2025). Here, we train SuperBPE tokenizers on top of the OLMo 2 tokenizer to reach vocabulary sizes of {200k, 400k} using the same 10GB text sample as Liu et al. (2025) for tokenizer training. We use FOCUS (Dobler and de Melo, 2023) to initialize the embeddings of the new superword tokens.

Results are shown in Figure 3. Through transfer to SuperBPE, we can speed up the subword-level LM while retaining performance to a large extent. However, at some vocabulary size threshold, the subword-level LM breaks off the frontier as the softmax begins to dominate the FLOPs (for OLMo 2 1B, this is somewhere between a vocabulary size of 200k and 400k tokens). Byte-level LMs do not suffer from the softmax bottleneck. This enables unboundedly increasing efficiency at a smooth dropoff in performance. Interestingly, BPE

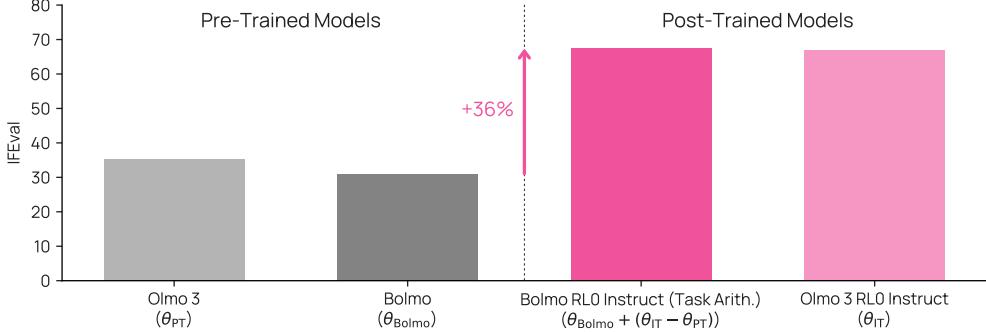


Figure 4 Byteified models can be post-trained by leveraging an existing (subword-level) post-trained Olmo 3 checkpoint; shown is the performance on IFEval of the base Olmo 3 model (θ_{PT}), the base Bolmo (θ_{Bolmo}), a post-trained Olmo 3 checkpoint (θ_{IT}), and the result of merging the post-trained checkpoint into Bolmo.

merges outperform entropy and cross-entropy merges, in contrast with prior work using entropy-based patch boundaries (Nawrot et al., 2023; Pagnoni et al., 2025). We believe this may be a pattern specific to the byteifying setting, since the BPE merging strategy is the one with the least amount of distinct merges to achieve any target compression (and thus, in this sense, the one closest to the pretrained model). Additional investigation with training from scratch would be necessary to validate this hypothesis.

5.2 Post-Training Byteified Models via Task Arithmetic

Takeaway. An existing subword-level post-trained checkpoint can be merged into a byteified model via Task Arithmetic (Ilharco et al., 2023) to post-train the byteified model with zero extra training cost.

Byteification adds a new component (a byteified model) to the ecosystem around the source LM. A natural question is: How does this new component interact with the other components of the source LM ecosystem? To answer this question, we investigate whether we can merge existing post-trained versions of Olmo 3 to post-train Bolmo without any extra training cost. We use the Olmo 3 checkpoint directly post-trained on instruction following via RL (RL-Zero; Olmo Team, 2025) in Deepseek-R1 style (DeepSeek-AI et al., 2025) as a case study. We find that we can infuse the instruction following capabilities from this checkpoint into Bolmo via Task Arithmetic (Ilharco et al., 2023) by adding the weight difference between the Transformer layers of the post-trained checkpoint and the base Olmo 3 to the corresponding Bolmo layers (see Figure 4).

While the Bolmo base model originally performs worse than Olmo 3 on IFEval (31.1% vs. 35.4%), merging via Task Arithmetic lifts performance to on par with the original post-trained checkpoint (67.4% vs. 66.9%). We conclude that it is possible to utilize components of the subword-level LM ecosystem to improve the corresponding byteified model. This removes the prerequisite for byte-level LM support in the infrastructure that subword-level LM post-training has benefitted immensely from (e.g., Lambert et al., 2024; Piché et al., 2025) and substantially speeds up iteration times.

A subtle requirement to post-training byteified models via Task Arithmetic is *embedding resettability*: since we only have a one-to-one correspondence between the parameters of the source subword-level LM and the parameters of the global model \mathcal{M} , we can only easily adapt \mathcal{M} via Task Arithmetic. The local encoder \mathcal{E} and decoder \mathcal{D} remain in the base model space. Whether the post-training transfer is successful thus depends on whether the base input embedding space (occupied by the local encoder \mathcal{E} and the input embedding matrix of the base model) and the base output embedding space (occupied by the local decoder \mathcal{D} and the output embedding matrix of the base model) remains compatible with post-trained inner Transformer layers; in other words, whether *resetting the embeddings of the post-trained model to the base model embeddings preserves performance*. We find this to be generally the case — and more so for larger models — although not always (see Appendix D). Designing post-training methods to preserve compatibility among components of the LM ecosystem is a promising area of research, with some encouraging early findings (e.g., Shenfeld et al., 2025).

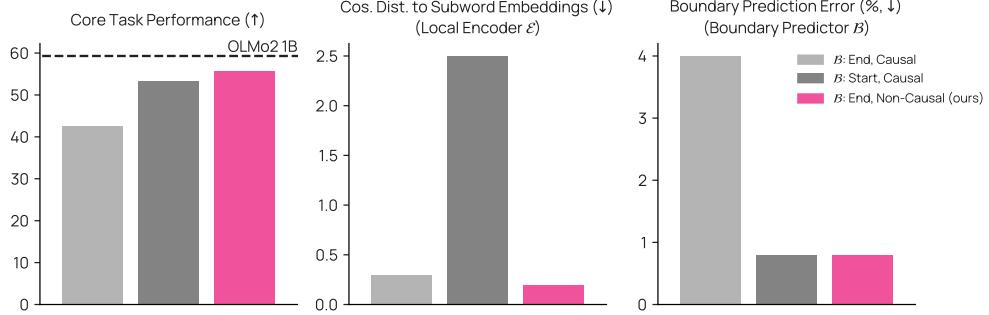


Figure 5 Boundary supervision by predicting the subword *patch start* or *patch end* using a *causal* or *non-causal* boundary predictor. Shown are the avg. task performance (*left*), cos. dist. of the local encoder representations to the target subword representations (*middle*), and the percentage of bytes where the predicted boundary differs from the true subword boundary (*right*) after Stage 1 training. Causal boundary predictors can achieve either accurate boundaries and accurate representations; non-causal boundaries enable both.

6 Ablations

6.1 Impact of Non-Causal Patch Boundaries

Takeaway. Causal boundary predictors have to choose between either matching the subword tokenizer boundaries or matching the subword patch content; non-causal boundary predictors can do both, substantially improving downstream performance.

Our largest deviation from prior byte-level architectures is non-causal boundary prediction. As per Section 3.1.1, causal boundary prediction suffers from a conundrum: we either predict the start of every subword patch, which is easy but creates an offset of one byte w.r.t. the patches passed to the original subword model while also making the patches less semantically coherent, or we predict the end of every subword patch, which is hard, especially since this task has to be performed by the shallow local encoder. In contrast, non-causal boundary prediction allows vastly simplifying the task by using future context (in our case, one future byte). This way, the shallow local encoder has enough capacity to perform well. Figure 5 quantifies this phenomenon: by predicting the patch end with future context, the patch end prediction task becomes easy, while retaining patches which are coherent and compatible with the global model. The remaining gap to the source subword-level LM is primarily caused by the non-causal boundary predictor still attaining less than 100% accuracy (see Appendix A for details); future work on designing the boundary predictor, potentially using more future context than a single byte, could close this gap. It would even be possible to retain the subword tokenizer for boundary prediction of the prefill. However, this would re-introduce reliance on an external tokenizer, add tokenization bias (see Section 2), and make training at higher compression factors (see Section 5.1) harder.

6.2 Is Stage 1 Training Necessary?

Takeaway. Stage 1 training improves performance, but is not strictly necessary to obtain a good final run. The key benefit of Stage 1 training is speeding up iteration times.

Training in two stages adds implementation complexity. Can we not just train everything end-to-end in a single stage instead and let the optimization process do the work? To address this question, we run experiments where we immediately train all parameters, initializing the local encoder, local decoder, boundary predictor and LM head randomly, and the parameters of the global model from the subword-level LM, i.e., starting directly from Stage 2. A fair comparison of Stage 2 only training with Stage 1 + Stage 2 training is difficult: Stage 1 training requires fewer FLOPs since we only backpropagate through a fraction of the global model (c.f.

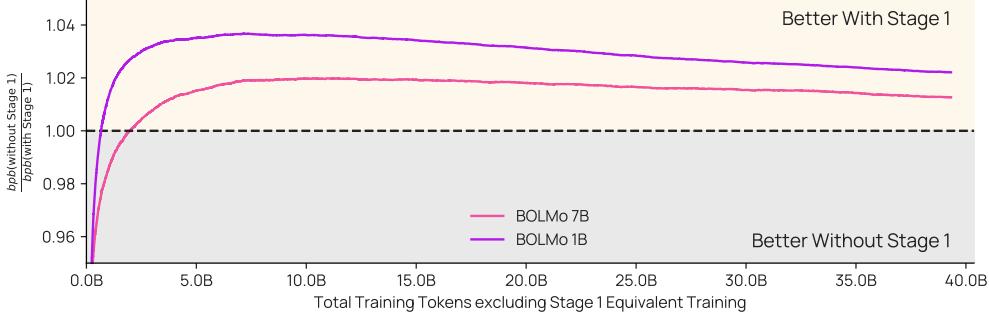


Figure 6 Ratio of bits-per-byte throughout training of runs without Stage 1 to runs with Stage 1. For runs with Stage 1, we exclude the Stage 1 loss trajectory. For runs without Stage 1, we exclude the first $9.8B \times 2/3 = 6.5B$ tokens, resulting in a comparable trajectory over the remaining $39.3B$ tokens; a ratio > 1 implies that Stage 1 is beneficial.

Section 3.2.1), and is more memory efficient since we only need to store a small fraction of the optimizer states by omitting training of the global model. We account for this difference by approximately FLOP-matching and disregarding the memory mismatch: Stage 1 needs approximately $2 \times \text{FLOPs}_{\mathcal{M}}$, whereas Stage 2 needs approximately $3 \times \text{FLOPs}_{\mathcal{M}}$ (1x for the forward and 2x for the backward pass through the global model). We thus add $9.8B \times 2/3 = 6.5B$ tokens to Stage 2 training when omitting Stage 1 (increasing the length of Stage 2 by 17%). In practice, we believe the factor of $2/3$ may slightly favor the Stage 2-only run since the memory requirements for Stage 1 are lower (permitting a larger batch size) and inference-specific optimizations could be used to speed up the forward pass of the subword-level LM used in Stage 1.

Figure 6 compares the training trajectory of runs with vs. without Stage 1 training. There are two main takeaways: (i) the 1B model benefits more from Stage 1 training than 7B, indicating that larger models may be more robust to catastrophic forgetting through large gradients at the start of training when starting directly with Stage 2, and (ii) the bits-per-byte gap narrows throughout the training trajectory but remains in favor of adding Stage 1; it is not clear how this behavior is influenced by the learning rate scheduling so we cannot easily extrapolate to higher token budgets. Since the absence of Stage 1 does not cause catastrophic degradation, we believe it is a reasonable hypothesis that Stage 1 training becomes less important with larger token budgets; however, this might be influenced in nontrivial ways by factors such as the choice of data mix.

Summarily, Stage 1 is beneficial in terms of improving performance compared to matched Stage 2-only training, but not strictly necessary. A key benefit of Stage 1 is streamlining experimentation: quickly obtaining a checkpoint which should come close to the subword-level LMs performance creates a substantially shorter feedback loop than repeatedly running full training experiments.

6.3 Selecting the Right Local Model Architecture for Fast Inference

Takeaway. FLOP-derivative measurements (total training/inference FLOPs or FLOPs/byte) are a suboptimal proxy for model efficiency. We recommend primarily using wallclock inference time measurements to guide byte-level LM architecture choices.

Previous work on byte-level LMs largely compares against subword-level LMs by matching the total amount of training or inference (i.e., prefill) FLOPs (e.g. Pagnoni et al., 2025) or FLOPs/byte (Hwang et al., 2025). This provides an incomplete picture. As observed by prior work (e.g. Ma et al., 2018), FLOPs do not necessarily correlate with inference speed; some sources of FLOPs are inherently more amenable to being computed efficiently on today’s hardware than others, and decoding in Transformers is typically memory bound. We thus largely used inference speed measurements to guide our choice of local model architecture. Figure 7 shows prefilling latency (time to first byte) and decoding throughput (bytes/s) measurements of our chosen architecture, as well as various candidate local model architectures we explored.

The chosen Bolmo architecture using mLSTM (Beck et al., 2025a) achieves competitive speeds at decoding ~ 125 bytes/s vs. ~ 150 bytes/s for the subword model at the same compression, and ~ 1 s to prefill 72K bytes

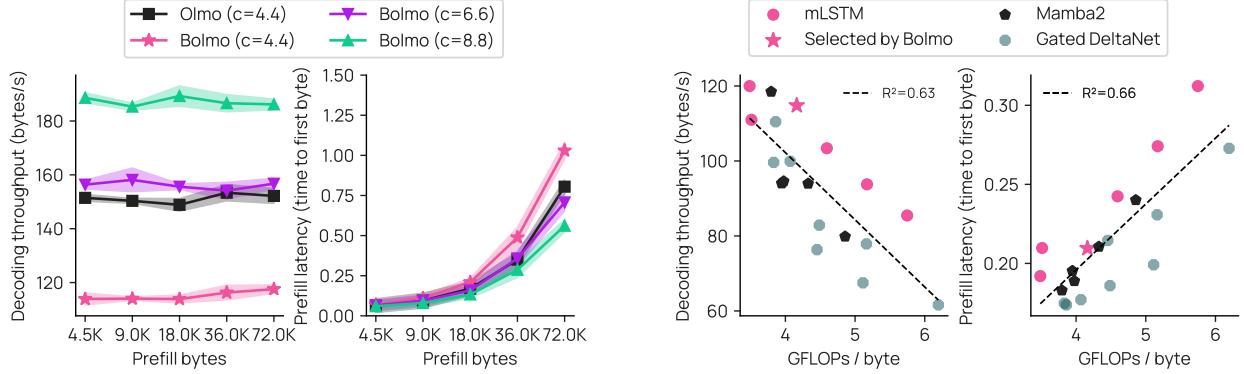


Figure 7 (left): decoding throughput (bytes/s) and prefilling latency (time to first byte) for Bolmo 7B and the source subword model across compression factors and prefill lengths; Bolmo 7B overtakes Olmo 3 7B at a compression of ~ 6.6 bytes per patch. (right): decoding throughput and prefilling latency for 18.0K prefill bytes across candidate local model architectures and of the final chosen Bolmo architecture. Recorded with batchsize=1 on H100 GPUs.

vs. ~ 0.8 s to prefill the tokens corresponding to the same number of bytes for the subword model. In addition, Bolmo can be made faster by training at arbitrarily higher compression factors (in contrast to subword-level LMs, see Section 5.1), and starts surpassing the subword model in inference efficiency at ~ 6.6 bytes per patch. As shown in Figure 7 (right), we find mLSTM as implemented in Tiled Flash Linear Attention (TFLA; Beck et al., 2025b) to achieve substantially higher wallclock decoding throughput than Mamba2 and Gated DeltaNet at the same amount of FLOPs/byte. Relying purely on FLOPs to guide architecture choices would have thus potentially resulted in suboptimal inference speed due to the inconsistent correlation between the two ($R^2 \approx 0.63$ to 0.66 in our experiments).

FLOP-matching is further complicated by having to make decisions as to how to count FLOPs, which is not trivial in practice. For example, the popular FLOP formulas from Hoffmann et al. (2022) assume a matrix multiplication of the input embeddings with the one-hot encoded input tokens. This is arguably not in line with hardware realities since the input embeddings can be computed via an extremely fast lookup operation, so counting the associated FLOPs can cause systematic biases.²⁰ Additionally, the chunk size used to partially parallelize linear RNN training inherently provides a way to use more FLOPs to achieve faster training (via higher parallelization; as in Dao and Gu, 2024; Yang et al., 2025), which further muddies the relationship between FLOPs and wallclock times.

7 Conclusion

We have introduced byteification as a missing additional direction to training from scratch for developing byte-level LMs. Byteification let us create Bolmo, the first fully open family of byte-level LMs on par with or surpassing state-of-the-art subword-level LMs at the 1B and 7B parameter scales. Bolmo benefits from architectural and training decisions specifically designed for byteifying, and comes close to matching subword-level LMs in inference speed. We have further explored byte-level models’ increased flexibility, such as arbitrarily decreasing token granularity for faster inference. Byteifying also lets us leverage other components of the ecosystem around the source subword model by byteifying post-trained models in zero-shot once the corresponding base model is byteified. Overall, byteifying finally makes byte-level LMs a practical choice competitive with subword-level LMs, and enables future research directions on byte-level LMs for both the byteification setting and training from scratch.

²⁰Counting the input embedding FLOPs has limited effect if the models being compared have similar vocabulary sizes. However, for example in the case of Hwang et al. (2025), it overestimates the FLOPs required by the subword-level baseline LM by up to $\sim 25\%$: The GPT3-Large matched Transformer baseline with $d = 1536$, $|V| = 128256$ and an average number of 4.6 bytes per patch is considered to require 0.42 GFLOPs/byte, of which $2 \times 1536 \times 128256 / 4.6 \approx 0.085$ GFLOPs/byte are due to the input embeddings, while a negligible amount of the GFLOPs/byte of the byte-level models are due to the input embeddings.

8 Future Directions

We believe Bolmo enables a number of future research directions, bits of which are sketched below.

Bit 0. Investigating how architectures optimized for byteification perform when training from scratch. We have restricted ourselves purely to the byteification setting. For example, we have not assessed how non-causal patch boundaries perform when training from scratch. We expect that the increased expressivity of the boundary predictor might be generally useful, but we do not yet know.

Bit 1. Learning non-causal boundaries end-to-end. We have purely trained our boundary predictor through direct external supervision — either to match subword tokens, or to match merges over subword tokens. We believe a highly promising area is learning non-causal boundary predictors end-to-end. For example, boundaries could be learnt end-to-end during post-training of a byteified model via RL, or by adapting methods like Hwang et al. (2025)’s method of enabling gradient flow through the boundary predictor to the non-causal setting.

Bit 2. Scaling patch size and local model capacity. We have designed the local models of Bolmo to minimize inference speed degradation when keeping the same patch size as the original subword model, since we have focused mainly on byteifying while keeping the patching constant. However, jointly using larger local models and a larger patch size might yield a better performance vs. efficiency tradeoff, as suggested by Pagnoni et al. (2025) and Huang et al. (2025).

Bit 3. Multi-byte prediction. While multi-token/byte prediction has been used to great effect to speed up language models (Gloeckle et al., 2024; Cai et al., 2024; Grivas et al., 2025, among others), Bolmo only predicts the direct next byte. It is not clear how many sequential invocations of the global model multi-byte prediction could save; however, even saving sequential local model computations could lead to substantial speedups and permit larger local models, synergizing with Bit 2.

Bit 4. Non-destructive byteification. As per Appendix A, the remaining gap between the performance of Bolmo and the original model can to a large extent be attributed to the continued training setup generally hurting performance. Investigating ways to make continued training less destructive, such as PEFT methods (e.g. Hu et al., 2022; Pfeiffer et al., 2023), could be promising.

Bit 5. Specialized LTLM sampling methods. Subword-level language models have benefitted from a range of sampling methods which have been to various extents designed for, or at the least empirically validated on, predominantly subword-level LMs (e.g. Holtzman et al., 2020; Meister et al., 2023; Minh et al., 2025). We have not investigated how these methods transfer to LTLMs. Developing specialized sampling methods for LTLMs, for instance by adjusting the sampling strategy based on the position of the current byte within the patch, is also an intriguing topic.

Bit 6. More equitable input units. Bolmo operates over UTF-8 bytes, which is a highly Latin-centric atomic unit (Limišiewicz et al., 2024). We believe that the dynamic latent tokenization can to some extent ‘amortize’ over the choice of the atomic unit, but it is not clear to what extent this is possible, and in how far LTLMs inherit the biases from their underlying encoding. Future work could investigate this, alongside alternative choices for the atomic unit such as MYTE (Limišiewicz et al., 2024) or SCRIPT (Land and Arnett, 2025).

Bit 7. Batched inference optimizations. We have shown that Bolmo can achieve throughputs competitive with subword-level LMs in the batchsize=1 setting, which is sufficient for edge applications. However, achieving fast batched inference of LTLMs by applying e.g. PagedAttention (Kwon et al., 2023) and continuous batching (Yu et al., 2022) will be necessary to unlock a wider range of applications. Here, there are some additional challenges for LTLMs caused by their dynamicity (a fixed amount of tokens across examples causes a variable number of bytes and vice versa) which require additional work.²¹

²¹To our knowledge, the only investigation into efficient batched LTLM inference so far is through Aleph Alpha’s vilm fork.

Acknowledgments

We thank the Beaker team at Ai2 for providing and maintaining the training infrastructure. We thank Tyler Romero for helpful discussions on inference efficiency, David Heineman for help with the evaluation infrastructure, Will Merill for useful discussions on linear RNNs, and Alisa Liu for useful discussions on tokenization. This work has been supported by the UK EPSRC grant EP/T02450X/1, and resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. Edoardo M. Ponti is supported by the ERC Starting Grant AToM-FM (101222956). We acknowledge the National Artificial Intelligence Research Resource (NAIRR) Pilot and Microsoft Azure for contributing to the results in this work.

References

- I. Athanasiadis, A. Karmush, and M. Felsberg. Model stitching by functional latent alignment. *arXiv preprint arXiv:2505.20142*, 2025.
- J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- M. Beck, K. Pöppel, P. Lippe, R. Kurle, P. M. Blies, G. Klambauer, S. Böck, and S. Hochreiter. xLSTM 7b: A recurrent LLM for fast and efficient inference. In *Forty-second International Conference on Machine Learning*, 2025a. URL <https://openreview.net/forum?id=LV3DpKD08B>.
- M. Beck, K. Pöppel, P. Lippe, and S. Hochreiter. Tiled Flash Linear Attention: More efficient linear rnn and xlstm kernels. *arXiv*, 2503.14376, 2025b. URL <https://arxiv.org/abs/2503.14376>.
- L. Beinborn and Y. Pinter. Analyzing cognitive plausibility of subword tokenization. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4478–4486, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.272. URL <https://aclanthology.org/2023.emnlp-main.272/>.
- A. Bick, K. Y. Li, E. P. Xing, J. Z. Kolter, and A. Gu. Transformers to ssms: Distilling quadratic knowledge to subquadratic models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 31788–31812. Curran Associates, Inc., 2024. doi: 10.5220/079017-0999. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/3848fef259495bfd04d60cdc5c1b4db7-Paper-Conference.pdf.
- Y. Bisk, R. Zellers, R. Le bras, J. Gao, and Y. Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020. doi: 10.1609/aaai.v34i05.6239. URL <https://ojs.aaai.org/index.php/AAAI/article/view/6239>.
- Z. Borsos, R. Marinier, D. Vincent, E. Kharitonov, O. Pietquin, M. Sharifi, D. Roblek, O. Teboul, D. Grangier, M. Tagliasacchi, et al. Audiolum: a language modeling approach to audio generation. *IEEE/ACM transactions on audio, speech, and language processing*, 31:2523–2533, 2023.
- T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv: 2401.10774*, 2024.
- F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, et al. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.
- M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillett, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. 2021.
- P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *CoRR*, arXiv:1803.05457, 2018.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- A. Cosma, S. Ruseti, E. Radoi, and M. Dascalu. The strawberry problem: Emergence of character-level understanding in tokenized language models, 2025. URL <https://arxiv.org/abs/2505.14172>.
- T. Dao and A. Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning*, 2024. URL <https://openreview.net/forum?id=ztn8FCR1td>.
- DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen,

- J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL <https://arxiv.org/pdf/2501.12948>.
- K. Dobler and G. de Melo. FOCUS: Effective embedding initialization for monolingual specialization of multilingual models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13440–13454, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.829. URL <https://aclanthology.org/2023.emnlp-main.829>.
- K. Dobler, D. Elliott, and G. de Melo. Token distillation: Attention-aware input embeddings for new tokens, 2025. URL <https://arxiv.org/abs/2505.20133>.
- A. Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In J. Burstein, C. Doran, and T. Solorio, editors, *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 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL <https://aclanthology.org/N19-1246>.
- L. Edman, H. Schmid, and A. Fraser. CUTE: Measuring LLMs’ understanding of their tokens. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3017–3026, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.177. URL <https://aclanthology.org/2024.emnlp-main.177>.
- L. Edman, H. Schmid, and A. Fraser. EXECUTE: A multilingual benchmark for LLM token understanding. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1878–1887, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.95. URL <https://aclanthology.org/2025.findings-acl.95>.
- D. Feher, I. Vulić, and B. Minixhofer. Retrofitting large language models with dynamic tokenization. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29866–29883, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1444. URL <https://aclanthology.org/2025.acl-long.1444>.
- W. Fleshman and B. V. Durme. Toucan: Token-aware character level language modeling, 2023. URL <https://arxiv.org/abs/2311.08620>.
- F. Gloeckle, B. Y. Idrissi, B. Roziere, D. Lopez-Paz, and G. Synnaeve. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning*, 2024. URL <https://openreview.net/forum?id=pEWAcejiU2>.
- A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- A. Grivas, L. Loconte, E. van Krieken, P. Nawrot, Y. Zhao, E. Wielewski, P. Minervini, E. Ponti, and A. Vergari. Fast and expressive multi-token prediction with probabilistic circuits, 2025. URL <https://arxiv.org/abs/2511.11346>.
- Y. Gu, O. Tafjord, B. Kuehl, D. Haddad, J. Dodge, and H. Hajishirzi. Olmes: A standard for language model evaluations. *ArXiv*, abs/2406.08446, 2024. URL <https://api.semanticscholar.org/CorpusID:270391754>.
- D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, et al. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.
- M. Haltiuk and A. Smywiński-Pohl. Model-aware tokenizer transfer, 2025. URL <https://arxiv.org/abs/2510.21954>.

- J. Hayase, A. Liu, N. A. Smith, and S. Oh. Sampling from your language model one byte at a time, 2025. URL <https://arxiv.org/abs/2506.14123>.
- D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- N. Ho, S. Bae, T. Kim, hyunjik.jo, Y. Kim, T. Schuster, A. Fisch, J. Thorne, and S.-Y. Yun. Block transformer: Global-to-local language modeling for fast inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=6osgTNnAZQ>.
- J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rygGQyrFvH>.
- E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeeFYf9>.
- H. Huang, D. Zhu, B. Wu, Y. Zeng, Y. Wang, Q. Min, and Z. Xun. Over-tokenized transformer: Vocabulary is generally worth scaling. In A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 26261–26282. PMLR, 13–19 Jul 2025. URL <https://proceedings.mlr.press/v267/huang25bb.html>.
- S. Hwang, B. Wang, and A. Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL <https://arxiv.org/abs/2507.07955>.
- G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=6t0Kwf8-jrj>.
- D. Jin, E. Pan, N. Oufattolle, W.-H. Weng, H. Fang, and P. Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.
- A. Kaushal and K. Mahowald. What do tokens know about their characters and how do they know it? In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2487–2507, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.179. URL <https://aclanthology.org/2022.naacl-main.179/>.
- T. Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In I. Gurevych and Y. Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL <https://aclanthology.org/P18-1007>.
- T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL <https://aclanthology.org/Q19-1026>.
- W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL <https://arxiv.org/abs/2309.06180>.
- Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-T. Yih, D. Fried, S. Wang, and T. Yu. Ds-1000: A natural and reliable benchmark for data science code generation. *ArXiv*, abs/2211.11501, 2022.
- N. Lambert, J. D. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, Y. Gu, S. Malik, V. Graf, J. D. Hwang, J. Yang, R. L. Bras, O. Tafjord, C. Wilhelm, L. Soldaini, N. A. Smith, Y. Wang, P. Dasigi, and H. Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. 2024. URL <https://api.semanticscholar.org/CorpusID:274192505>.
- A. Łaniczcki, K. Staniszewski, P. Nawrot, and E. Ponti. Inference-time hyper-scaling with KV cache compression. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=8ZiElzQxf1>.

- S. Land and C. Arnett. Bpe stays on script: Structured encoding for robust multilingual pretokenization, 2025. URL <https://arxiv.org/abs/2505.24689>.
- A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.
- D. Liang, H. Gonen, Y. Mao, R. Hou, N. Goyal, M. Ghazvininejad, L. Zettlemoyer, and M. Khabsa. XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13142–13152, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.813. URL <https://aclanthology.org/2023.emnlp-main.813/>.
- T. Limisiewicz, T. Blevins, H. Gonen, O. Ahia, and L. Zettlemoyer. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15059–15076, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.804. URL <https://aclanthology.org/2024.acl-long.804/>.
- A. Liu, J. Hayase, V. Hofmann, S. Oh, N. A. Smith, and Y. Choi. Superbpe: Space travel for language models, 2025. URL <https://arxiv.org/abs/2503.13423>.
- J. Lotz, E. Salesky, P. Rust, and D. Elliott. Text rendering strategies for pixel language models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10155–10172, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.628. URL <https://aclanthology.org/2023.emnlp-main.628/>.
- N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018. URL <https://arxiv.org/abs/1807.11164>.
- C. Meister, T. Pimentel, G. Wiher, and R. Cotterell. Locally typical sampling. *Transactions of the Association for Computational Linguistics*, 11:102–121, 2023. doi: 10.1162/tacl_a_00536. URL <https://aclanthology.org/2023.tacl-1.7/>.
- S. J. Mielke, Z. Alyafeai, E. Salesky, C. Raffel, M. Dey, M. Gallé, A. Raja, C. Si, W. Y. Lee, B. Sagot, et al. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*, 2021.
- N. N. Minh, A. Baker, C. Neo, A. G. Roush, A. Kirsch, and R. Shwartz-Ziv. Turning up the heat: Min-p sampling for creative and coherent LLM outputs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=FBkpCyuJts>.
- B. Minixhofer, F. Paischer, and N. Rekabsaz. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3992–4006, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.293. URL <https://aclanthology.org/2022.naacl-main.293/>.
- B. Minixhofer, J. Pfeiffer, and I. Vulić. CompoundPiece: Evaluating and improving decompounding performance of language models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 343–359, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.24. URL <https://aclanthology.org/2023.emnlp-main.24/>.
- B. Minixhofer, E. M. Ponti, and I. Vulić. Zero-shot tokenizer transfer, 2025a. URL <https://arxiv.org/abs/2405.07883>.
- B. Minixhofer, I. Vulić, and E. Ponti. Universal cross-tokenizer distillation via approximate likelihood matching. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL <https://openreview.net/forum?id=DxKP2E0xK2>.
- MosaicML. Llm foundry - jeopardy dataset. https://github.com/mosaicml/llm-foundry/blob/main/scripts/eval/local_data/world_knowledge/jeopardy_all.jsonl, 2024. Accessed: 2024-11-10.
- P. Nawrot, S. Tworkowski, M. Tyrolski, L. Kaiser, Y. Wu, C. Szegedy, and H. Michalewski. Hierarchical transformers are more efficient language models. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1559–1571, Seattle, United States, July 2022.

Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.117. URL <https://aclanthology.org/2022.findings-naacl.117/>.

- P. Nawrot, J. Chorowski, A. Lancucki, and E. M. Ponti. Efficient transformers with dynamic token pooling. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL <https://aclanthology.org/2023.acl-long.353/>.
- P. Neitemeier, B. Deisereth, C. Eichenberg, and L. Balles. Hierarchical autoregressive transformers: Combining byte- and word-level processing for robust, adaptable language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=tU074jg2vS>.
- T. OLMo, P. Walsh, L. Soldaini, D. Groeneveld, K. Lo, S. Arora, A. Bhagia, Y. Gu, S. Huang, M. Jordan, N. Lambert, D. Schwenk, O. Tafjord, T. Anderson, D. Atkinson, F. Brahman, C. Clark, P. Dasigi, N. Dziri, M. Guerquin, H. Ivison, P. W. Koh, J. Liu, S. Malik, W. Merrill, L. J. V. Miranda, J. Morrison, T. Murray, C. Nam, V. Pyatkin, A. Rangapur, M. Schmitz, S. Skjonsberg, D. Wadden, C. Wilhelm, M. Wilson, L. Zettlemoyer, A. Farhadi, N. A. Smith, and H. Hajishirzi. 2 olmo 2 furious, 2024. URL <https://arxiv.org/abs/2501.00656>.
- T. OLMo, P. Walsh, L. Soldaini, D. Groeneveld, K. Lo, S. Arora, A. Bhagia, Y. Gu, S. Huang, M. Jordan, N. Lambert, D. Schwenk, O. Tafjord, T. Anderson, D. Atkinson, F. Brahman, C. Clark, P. Dasigi, N. Dziri, A. Ettinger, M. Guerquin, D. Heineman, H. Ivison, P. W. Koh, J. Liu, S. Malik, W. Merrill, L. J. V. Miranda, J. Morrison, T. Murray, C. Nam, J. Poznanski, V. Pyatkin, A. Rangapur, M. Schmitz, S. Skjonsberg, D. Wadden, C. Wilhelm, M. Wilson, L. Zettlemoyer, A. Farhadi, N. A. Smith, and H. Hajishirzi. 2 olmo 2 furious, 2025. URL <https://arxiv.org/abs/2501.00656>.
- Olmo Team. Olmo 3, 2025. URL <https://allenai.org/papers/olmo3>.
- A. Pagnoni, R. Pasunuru, P. Rodriguez, J. Nguyen, B. Muller, M. Li, C. Zhou, L. Yu, J. E. Weston, L. Zettlemoyer, G. Ghosh, M. Lewis, A. Holtzman, and S. Iyer. Byte latent transformer: Patches scale better than tokens. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.453. URL [https://aclanthology.org/2025.acl-long.453/](https://aclanthology.org/2025.acl-long.453).
- A. Pal, L. K. Umapathi, and M. Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In G. Flores, G. H. Chen, T. Pollard, J. C. Ho, and T. Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR, 07–08 Apr 2022. URL <https://proceedings.mlr.press/v174/pal22a.html>.
- D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.
- J. Pfeiffer, S. Ruder, I. Vulić, and E. Ponti. Modular deep learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL <https://openreview.net/forum?id=z9EkXfvxta>. Survey Certification.
- B. Phan, B. Amos, I. Gat, M. Havasi, M. Muckley, and K. Ullrich. Exact byte-level probabilities from tokenized language models for fim-tasks and model ensembles. *arXiv preprint arXiv:2410.09303*, 2024.
- A. Piché, E. Kamalloo, R. Pardinas, X. Chen, and D. Bahdanau. Pipelinrl: Faster on-policy reinforcement learning for long sequence generation, 2025. URL <https://arxiv.org/abs/2509.19128>.
- P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In J. Su, K. Duh, and X. Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL <https://aclanthology.org/D16-1264>.
- S. Reddy, D. Chen, and C. D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL <https://aclanthology.org/Q19-1016>.
- P. Rust, J. F. Lotz, E. Bugliarello, E. Salesky, M. de Lhoneux, and D. Elliott. Language modelling with pixels. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=FkSp8VW8RjH>.
- K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial winograd schema challenge at

scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, Apr. 2020. doi: 10.1609/aaai.v34i05.6399. URL <https://ojs.aaai.org/index.php/AAAI/article/view/6399>.

- M. Sap, H. Rashkin, D. Chen, R. Le Bras, and Y. Choi. Social IQa: Commonsense reasoning about social interactions. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL <https://aclanthology.org/D19-1454>.
- R. Sennrich, B. Haddow, and A. Birch. Neural Machine Translation of Rare Words with Subword Units. In K. Erk and N. A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL <https://aclanthology.org/P16-1162>.
- I. Shenfeld, J. Pari, and P. Agrawal. RL’s razor: Why online reinforcement learning forgets less, 2025. URL <https://arxiv.org/abs/2509.04259>.
- K. Slagle. Spacebyte: Towards deleting tokenization from large language modeling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=KEe4IUUp201>.
- A. Talmor, J. Herzig, N. Lourie, and J. Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In J. Burstein, C. Doran, and T. Solorio, editors, *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 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL <https://aclanthology.org/N19-1421>.
- D. Tito Svenstrup, J. Hansen, and O. Winther. Hash embeddings for efficient word representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f0f6ba4b5e0000340312d33c212c3ae8-Paper.pdf.
- K. Tran. From english to foreign languages: Transferring pre-trained language models, 2020. URL <https://arxiv.org/abs/2002.07306>.
- O. Uzan and Y. Pinter. Charbench: Evaluating the role of tokenization in character-level tasks, 2025. URL <https://arxiv.org/abs/2508.02591>.
- T. Vieira, B. LeBrun, M. Julianelli, J. L. Gastaldi, B. DuSell, J. Terilla, T. J. O’Donnell, and R. Cotterell. From language models over tokens to language models over characters. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=sQSOr0nQZR>.
- J. Wang, T. Gangavarapu, J. N. Yan, and A. M. Rush. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=X1xNsukssb>.
- H. Wei, Y. Sun, and Y. Li. Deepseek-ocr: Contexts optical compression, 2025. URL <https://arxiv.org/abs/2510.18234>.
- J. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions. In L. Derczynski, W. Xu, A. Ritter, and T. Baldwin, editors, *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL <https://aclanthology.org/W17-4413/>.
- L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10: 291–306, 2022. doi: 10.1162/tacl_a_00461. URL <https://aclanthology.org/2022.tacl-1.17/>.
- S. Yang, J. Kautz, and A. Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=r8H7xhYPwz>.
- G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL <https://www.usenix.org/conference/osdi22/presentation/yu>.
- L. Yu, D. Simig, C. Flaherty, A. Aghajanyan, L. Zettlemoyer, and M. Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=JTm02V9Xpz>.

- R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In A. Korhonen, D. Traum, and L. Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL <https://aclanthology.org/P19-1472>.
- Y. Zhang and Q. Yang. A survey on multi-task learning, 2021. URL <https://arxiv.org/abs/1707.08114>.
- L. Zheng, X. Zhao, G. Wang, C. Wu, D. Dong, A. Wang, M. Wang, Y. Du, H. Bo, A. Sharma, B. Li, K. Zhang, C. Hu, U. Thakker, and L. Kong. Evabyte: Efficient byte-level language models at scale, 2025. URL <https://hkunlp.github.io/blog/2025/evabyte>.
- T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widayarsi, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

	\mathcal{E}	Sim.	\mathcal{B}	Acc.	L/G	ARC	MMLU	CSQA	HS	WinoG	SocialIQAPIQA	B.Skills	CUTE	Avg.
OLMo2 1B	-	-	-	-	61.4	40.4	66.0	68.9	65.2	55.1	76.4	72.9	27.5	59.3
Oracle Boundaries ($\mathcal{B} = \mathcal{B}_{\text{Subword}}$)														
Start, C	97.5	100	8.8	57.4	35.4	64.4	65.0	65.1	54.0	74.5	66.2	19.6	55.7	
End, C	99.7	100	8.8	59.0	36.7	64.1	67.4	65.1	53.2	74.6	71.3	30.8	58.0	
End, NC(S)	99.8	100	9.8	58.5	36.8	63.4	68.0	66.1	53.6	74.6	71.6	31.2	58.2	
End, NC(F)	99.8	100	8.8	58.2	36.8	62.6	67.7	66.9	52.4	75.0	70.7	30.7	57.9	
Learned Boundary Prediction ($\mathcal{B} \in \{\mathcal{B}_{\text{H-Net}}, \mathcal{B}_{\text{BOLMo}}\}$)														
Start, C	97.5	99.2	8.8	54.1	33.4	63.6	57.9	64.1	52.9	70.4	64.7	19.6	53.4	
End, C	99.7	96.0	8.8	45.9	29.3	43.4	41.8	57.5	44.0	62.4	50.8	7.8	42.5	
End, NC(S)	99.8	99.2	9.8	56.2	34.7	61.3	61.4	64.2	52.4	71.6	69.3	29.8	55.7	
End, NC(F)	99.8	99.2	8.8	56.2	34.6	61.9	61.4	65.0	51.7	71.9	69.0	28.8	55.6	

Table 3 Comparison of various boundary prediction settings after Stage 1 training across oracle (subword) boundaries and the boundaries as predicted, predicting the patch start causally (**Start, C**), patch end causally (**End, C**), patch end non-causally using a separate boundary symbol (**End, NC(S)**) and patch end non-causally with fused boundaries (**End, NC(F)**), our chosen setting. \mathcal{E} Sim. = the cosine similarity of the pooled local encoder representations to the corresponding subword embeddings, \mathcal{B} Acc.= accuracy of the boundary predictor. L/G = average number of local model invocations per global model invocation. **Boldface** indicates the best result per column.

A Additional Ablations

Analyzing the choice of boundary predictor. Table 3 compares various choices for the boundary predictor, confirming that fused non-causal boundary prediction of the patch end is best for bytefying. Additionally, analyzing the performance under the original subword ('oracle') boundaries shows that the remainder of the gap to the source model after Stage 1 can be mostly explained by the remaining small percentage of errors of the boundary predictor.

Comparing byteification to standard continued training. Table 4 compares Bolmo to an Olmo 3 model with continued training on the same data under the same training settings (same batch size, optimizer, etc., see Table 8). Continued training without byteification generally degrades performance, potentially forgetting due to a narrower data mix and suboptimal training procedure. A notable exception is character understanding, where the model improves due to the training data targeting this skill (Appendix C), but remains worse than Bolmo. While some gap between the bytefied model and the model with continued training persists, we believe a promising direction to improve bytefying is thus to apply techniques which generally make training less prone to forgetting, such as applying PEFT methods (e.g. Hu et al., 2022; Pfeiffer et al., 2023).

B Benchmark Details

We utilize OLMES (OLMo et al., 2024) for all evaluations. See Table 5 for details on our 7B evaluation suite, and Table 6 for details on our 1B evaluation suite.

C CUTE-Style Training Data

To encourage models trained on our data mix to learn information about the characters within a word, we generate $\sim 75M$ tokens ($\sim 0.04\%$ of the training data) of tasks requiring character-level understanding using the CUTE repository. Tasks include spelling out words, reversing words as well as swapping, deleting, and substituting characters within a word given words in a source wordlist. We use a list of $n = 150000$ words, ensuring zero overlap with the CUTE test words to avoid contamination. This data is purely in English. We do not use any multilingual character understanding data, but still observe large improvements on the multilingual EXECUTE benchmark (see Section 5), suggesting that some texts requiring character-level understanding can help acquire generalizable knowledge about the characters within words. We observed

	Bolmo 7B	Olmo 3 CT 7B	Olmo 3 7B
Char	75.1	<u>71.0</u>	56.0
CUTE	78.6	<u>72.9</u>	56.9
EXECUTE	71.6	<u>69.2</u>	55.1
Code	40.7	37.8	<u>39.5</u>
HumanEval pass@1/@16	40.6 / 74.7	<u>42.4</u> / 64.9	49.0 / <u>71.1</u>
DeepSeek LeetCode pass@1/@16	2.3 / 7.6	0.7 / 2.9	<u>1.6</u> / 6.2
DS 1000	14.9	<u>19.4</u>	20.1
MBPP pass@1/@16	42.8 / 68.0	45.7 / <u>57.2</u>	<u>44.3</u> / 54.9
MultiPL HumanEval pass@1/@16	26.8 / 62.5	<u>30.1</u> / 53.4	33.6 / <u>56.3</u>
MultiPL MBPP pass@1/@16	<u>38.0</u> / 69.2	38.4 / <u>60.5</u>	37.8 / 59.9
Math	48.9	<u>49.8</u>	55.3
GSM8K	<u>68.0</u>	67.6	73.1
MATH	29.8	<u>32.0</u>	37.5
MC STEM	65.5	<u>66.1</u>	66.3
ARC MC	88.5	<u>88.7</u>	89.2
MMLU STEM	57.0	<u>57.7</u>	59.5
MedMCQA MC	47.8	48.9	<u>48.2</u>
MedQA MC	<u>42.4</u>	42.9	42.0
SciQ MC	91.9	<u>92.5</u>	92.8
MC Non-STEM	75.8	<u>76.6</u>	77.7
MMLU Humanities	67.2	<u>68.2</u>	69.2
MMLU Social Sci.	<u>74.0</u>	75.2	75.2
MMLU Other	65.1	<u>66.2</u>	66.9
CSQA MC	73.6	<u>74.6</u>	75.2
PiQA MC	<u>79.4</u>	79.1	80.3
SocialIQA MC	79.1	<u>79.2</u>	80.4
CoQA Gen2MC MC	90.0	<u>90.2</u>	92.9
DROP Gen2MC MC	59.1	<u>61.1</u>	62.5
Jeopardy Gen2MC MC	84.8	86.3	<u>85.5</u>
NaturalQs Gen2MC MC	65.9	<u>66.6</u>	69.6
SQuAD Gen2MC MC	<u>95.8</u>	95.7	96.8
GenQA	70.9	<u>71.7</u>	72.4
HellaSwag RC	78.8	<u>78.6</u>	77.8
Winogrande RC	85.5	85.8	<u>85.7</u>
Lambada	71.1	<u>69.9</u>	68.0
Basic Skills	89.6	<u>89.8</u>	90.0
DROP	<u>65.2</u>	65.0	71.5
Jeopardy	56.8	63.1	<u>60.3</u>
NaturalQs	28.6	<u>31.1</u>	32.6
SQuAD	91.6	<u>92.0</u>	93.5
CoQA	70.5	70.0	72.7

Table 4 Results comparing Bolmo to the source Olmo 3 model with continued training for the same amount of tokens on the Bolmo data mix and the original Olmo 3; the degradation from byteifying is not specifically caused by the conversion to bytes; it can to a substantial extent be attributed to continued training of the source model. **Boldface** indicates the best result per task, underline the second best.

that byte-level models otherwise do not acquire this knowledge through our short training schedule. However, training for longer, on more diverse data, or with larger local models could act as alternative routes to acquire character-level knowledge.

D Does Post-Training Byteified Models via Task Arithmetic Always Work?

As outlined in Section 5.2, embedding resetability is a crucial prerequisite of post-training byteified models via Task Arithmetic. This is the case since we can transfer the global model \mathcal{M} to the post-trained space via

	task	ICL	format	metric	temp	top-p	max toks	p@k (n)	# sub
Bolmo 7B Suite									
<i>Char</i>	CUTE (2024)	4	Greedy Cont.	Acc	-	-	-	-	-
	EXECUTE (2025)	4	Greedy Cont.	Acc	-	-	-	-	-
<i>Code</i>	HumanEval (2021)	3	Code Exec.	pass@k	0.6	0.6	512	1, 16 (32)	-
	MBPP (2021)	3	Code Exec.	pass@k	0.6	0.6	512	1, 16 (32)	-
	BigCodeBench (2024)	3	Code Exec.	pass@k	0.6	0.6	1280	1 (5)	-
	DS 1000 (2022)	3	Code Exec.	pass@k	0.6	0.6	1024	1 (5)	-
	Deepseek LeetCode (2024)	0	Code Exec.	pass@k	0.6	0.6	512	1, 16 (32)	-
	MultiPL-E HumanEval (2022)	0	Code Exec.	pass@k	0.6	0.6	1024	1, 16 (32)	6
	MultiPL-E MBPP (2022)	0	Code Exec.	pass@k	0.6	0.6	1024	1, 16 (32)	6
<i>Math</i>	GSM8K (2021)	8 ^α	CoT EM	pass@k	0.6	0.6	512	1, 4 (8)	-
	Minerva MATH (2022)	4 ^α	CoT EM	pass@k	0.6	0.6	1024	1, 4 (4)	7
<i>STEM QA</i>	ARC (2018)	5	MC	Acc	-	-	-	-	2
	MMLU STEM (2021)	5	MC	Acc	-	-	-	-	19
	MedMCQA (2022)	5	MC	Acc	-	-	-	-	-
	MedQA (2021)	5	MC	Acc	-	-	-	-	-
	SciQ (2017)	5	MC	Acc	-	-	-	-	-
<i>Non-STEM QA</i>	MMLU Humanities (2021)	5	MC	Acc	-	-	-	-	13
	MMLU Social Sci. (2021)	5	MC	Acc	-	-	-	-	12
	MMLU Other (2021)	5	MC	Acc	-	-	-	-	14
	CSQA (2019)	5	MC	Acc	-	-	-	-	-
	PiQA (2020)	5	MC	Acc	-	-	-	-	-
	SocialIQA (2019)	5	MC	Acc	-	-	-	-	-
	DROP Gen2MC (Olmo Team (2025); 2019)	5	MC	Acc	-	-	-	-	-
<i>GenQA</i>	Jeopardy Gen2MC (Olmo Team (2025); 2024)	5	MC	Acc	-	-	-	-	-
	NaturalQs Gen2MC (Olmo Team (2025); 2019)	5	MC	Acc	-	-	-	-	-
	SQuAD Gen2MC (Olmo Team (2025); 2016)	5	MC	Acc	-	-	-	-	-
	CoQA Gen2MC (Olmo Team (2025); 2019)	0 [†]	MC	Acc	-	-	-	-	-
	Basic Skills (Olmo Team (2025))	5	MC	Acc	-	-	-	-	6
	HellaSwag (2019)	5	RC _{per-char}	Acc	-	-	-	-	-
	WinoGrande (2020)	5	RC _{none}	Acc	-	-	-	-	-
<i>GenQA</i>	Lambada (2016)	0	Greedy Cont.	Acc	-	-	-	-	-
	Basic Skills (Olmo Team (2025))	5	RC _{per-token}	Acc	-	-	-	-	6
	DROP (2019)	5	GenQA	F1	0	1	100	-	-
	Jeopardy (2024)	5	GenQA	F1	0	1	50	-	-
	NaturalQs (2019)	5	GenQA	F1	0	1	50	-	-
	SQuAD (2016)	5	GenQA	F1	0	1	50	-	-
	CoQA (2019)	0 [†]	GenQA	F1	0	1	50	-	-

Table 5 Details of the Bolmo 7B evaluation suite, adapted from Olmo Team (2025)’s OLMOBASEEVAL. Tasks were formatted as multiple-choice (MC), rank choice (RC, following the setup in Gu et al. (2024)), short-form generative (GenQA), chain-of-thought with exact-match scoring (CoT EM) or Code Execution (Code Exec.). [†] = few-shot examples are built-in the task; ^α = human-written few-shot examples; # sub = number of subtasks.

Task Arithmetic, but we can not transfer the local models since they do not have corresponding parameters in the post-trained checkpoint.

In Figure 8, we analyze embedding resetability across a number of models. Resetting the embeddings is possible without substantial performance degradation for a substantial fraction of the analyzed models, with a weak trend toward larger models being more amenable to it. Additionally, in line with the findings of Shenfeld et al. (2025), we find models post-trained via RL (the Olmo 3 RL-Zero family; Olmo Team, 2025) to be closer to the original model; here, embedding resetting almost perfectly preserves the original models’ performance.

Future work could investigate in more detail when post-training via Task Arithmetic is possible, and whether it is possible to restore the ability to byteify without additional training for post-trained models where this is not the case as-is.

E Embedding Rank Analysis

Figure 9 shows the explained variance ratio of the singular values of the input and output embedding matrices across a number of models. Besides one exception (Qwen3-4B-Base input embeddings), there is a substantial

task	capability	ICL	metric	# sub
Bolmo 1B Suite				
ARC*	Science QA	5	Acc	2
MMLU*	General QA	5	Acc	57
CSQA*	Commonsense QA	5	Acc	-
HellaSwag*	Language Modeling	5	Acc	-
WinoGrande*	Language Modeling	5	Acc	-
SocialIQA*	Social QA	5	Acc	-
PiQA*	Physical QA	5	Acc	-
CoQA	Conversation QA	0 [†]	Acc	-
DROP	Passage QA	5	Acc	-
Jeopardy	Trivia QA	5	Acc	-
NaturalQs	General QA	5	Acc	-
SQuAD	General QA	5	Acc	-
SciQ	Science QA	5	Acc	-
QASPER	Science QA	5	Acc	-
Basic Skills*	Basic QA	5	Acc	6
DBQA	Science QA	5	Acc	-
ProtocolQA	Science QA	5	Acc	-
Lambada	Language Modeling	0	Acc	-
MedMCQA	Medical QA	5	Acc	-
MedQA	Medical QA	5	Acc	-
SciRIFF	Science QA	5	Acc	-
CUTE*	Character Understanding	4	Acc	-

Table 6 Details of the Bolmo 1B evaluation suite, adapted from Olmo Team (2025)’s Base Easy Suite. Tasks were formatted as rank choice (RC, following the setup in Gu et al. (2024)) [†] = few-shot examples are built-in the task; ^α = human-written few-shot examples; # sub = number of subtasks, * = selected core tasks.

amount of high-rank structure. This makes the embeddings difficult to approximate using lower-dimensional local models. In particular, in the case of the local encoder, the embedding rank has a hard limit given by the dimensionality of the local model if a linear upprojection or padding is used to upproject (as done, e.g., by Hwang et al., 2025). Concatenation of the local representations, as done by Pagnoni et al. (2025), does not lead to a hard limit on the rank but may still limit expressivity.

F Full Hyperparameters

Full architecture hyperparameters are shown in Table 7, and full training in Table 8.

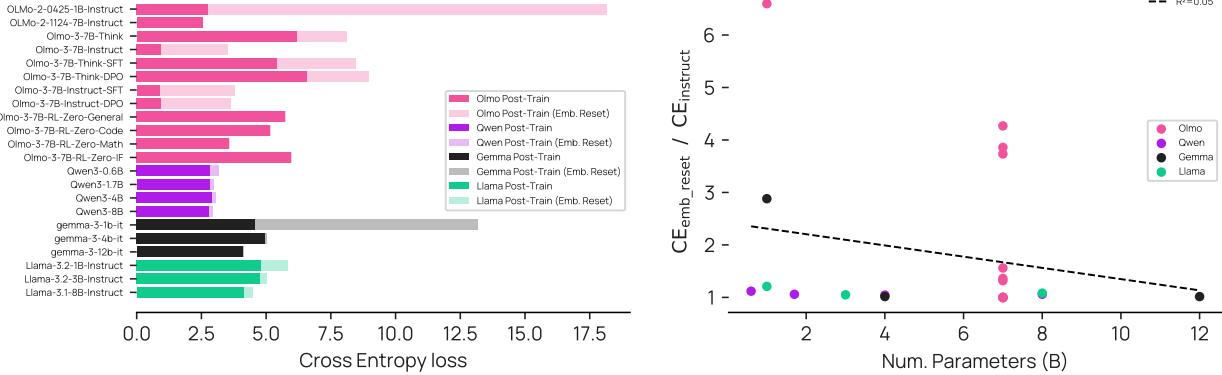


Figure 8 (left): Cross-Entropy loss for post-trained models, and the same post-trained models with the embeddings reset to the corresponding base model embeddings; loss is computed on examples from the Tulu 3 dataset (Lambert et al., 2024). (right): Number of model parameters vs. the loss ratio of the model with reset embeddings to the original post-trained model. The number of parameters explains some variance (with larger models being more amenable to reset embeddings), with the remaining variance presumably being due to different post-training choices.

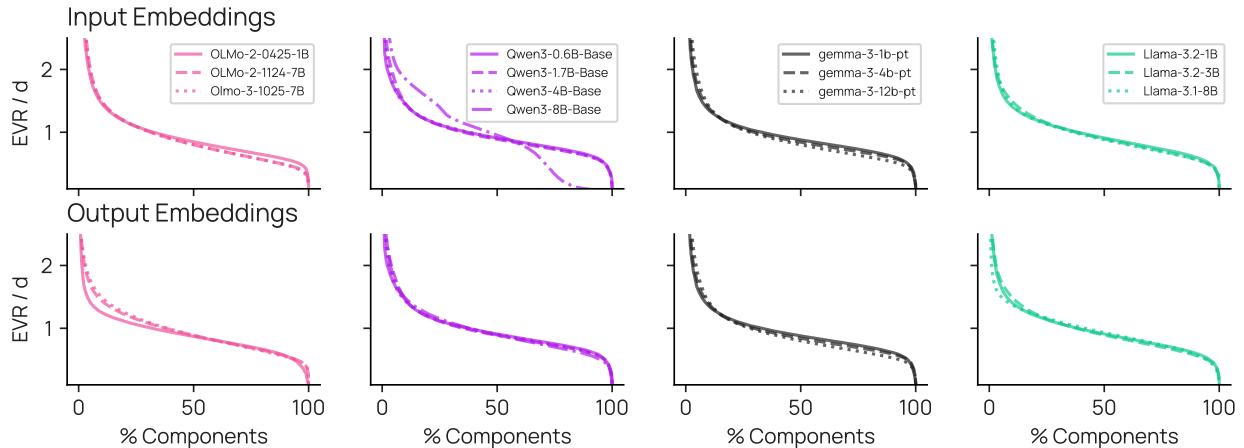


Figure 9 The explained variance ratio of the singular values of the input and output embedding matrices (normalized by the number of dimensions). The explained variance ratio smoothly decays along the number of components, up until a steep dropoff toward the highest-rank components. This indicates that it is difficult to approximate the embedding matrices using lower-rank structure. A notable exception is Qwen3-4B-Base, which may be more amenable to a lower-dimensional local encoder; we are not so bold as to dare a guess why.

	Bolmo 7B	Bolmo 1B
Global Model		
	Same as Olmo Team (2025)	Same as OLMo et al. (2024)
Local Encoder		
Dimension	4096	2048
Layer Type	mLSTM + FFN	mLSTM + FFN
Num. Layers	1	1
mLSTM		
Num. Heads	16	16
Nonlinearity	Exponential	Exponential
QK Dim.	128	128
V Dim.	256	256
Gate Soft Cap	15	15
Input Gate Bias Init.	-10	-10
FFN		
Expansion Dim.	5504	2816
Nonlinearity	SwiGLU	SwiGLU
Layer norm	RMSNorm	RMSNorm
Local Decoder		
Dimension	4096	2048
Layer Type	mLSTM + FFN	mLSTM + FFN
Num. Layers	4	4
mLSTM		
Num. Heads	16	16
Nonlinearity	Exponential	Exponential
QK Dim.	128	128
V Dim.	256	256
Gate Soft Cap	15	15
Input Gate Bias Init.	-10	-10
FFN		
Expansion Dim.	5504	2816
Nonlinearity	SwiGLU	SwiGLU
Layer norm	RMSNorm	RMSNorm

Table 7 Bolmo architecture details.

	Bolmo 7B	Bolmo 1B
Stage 1		
Total Training Tokens	9.8B	9.8B
Total Training Bytes	~43.1B	~43.1B
Training Steps	75K	75K
Batch Size	32	32
Max. Length. (Tokens)	4096	4096
Max. Length. (Bytes)	24576	24576
LR Schedule	Warmup + Linear	Warmup + Linear
Peak LR	5e-4	7e-4
Warmup Steps	7.5K	7.5K
Optimizer	AdamW	AdamW
Weight Decay	0.1	0.1
β_1, β_2	0.9, 0.95	0.9, 0.95
Max. Grad. Norm	0.5	0.5
Throughput (TPS)	9.9K	34.5K
Throughput (BPS)	59.4K	207K
Stage 2		
Total Training Tokens	39.3B	39.3B
Total Training Bytes	~172.9B	~172.9B
Training Steps	150K	150K
Batch Size	64	64
Max. Length. (Tokens)	4096	4096
Max. Length. (Bytes)	24576	24576
LR Schedule	Warmup + Linear	Warmup + Linear
Peak LR (Global Model)	1.8e-5	2.6e-5
Peak LR (Local Models)	3.7e-5	5.2e-5
Warmup Steps	15K	15K
Optimizer	AdamW	AdamW
Weight Decay	0.1	0.1
β_1, β_2	0.9, 0.95	0.9, 0.95
Max. Grad. Norm	0.5	0.5
Throughput (TPS)	6.3K	27.7K
Throughput (BPS)	37.8K	166.2K

Table 8 Bolmo training details. Throughput estimates are in tokens per second (TPS) and bytes per second (BPS) per accelerator, as achieved by the final training runs on H100 GPUs.