# From Bytes to Ideas: Language Modeling with Autoregressive U-Nets

**Mathurin Videau*** Meta AI

**Badr Youbi Idrissi*** Meta AI

**Alessandro Leite** INSA Rouen Normandy, LITIS

**Marc Schoenauer** TAU, Inria

**Olivier Teytaud** Thales - CortAIx-Labs

**David Lopez-Paz** Meta AI

## Abstract

Tokenization imposes a fixed granularity on the input text, freezing how a language model operates on data and how far in the future it predicts. Byte Pair Encoding (BPE) and similar schemes split text once, build a static vocabulary, and leave the model stuck with that choice. We relax this rigidity by introducing an autoregressive U-Net that learns to embed its own tokens as it trains. The network reads raw bytes, pools them into words, then pairs of words, then up to 4 words, yielding a multi-scale representation of the sequence. At deeper stages, the model must predict further into the future — anticipating the next few words rather than the next byte — so deeper stages focus on broader semantic patterns while earlier stages handle fine details. When carefully tuning and controlling pretraining compute, shallow hierarchies are on par with strong BPE baselines, and deeper hierarchies exhibit a promising trend. Because tokenization now lives inside the model, the same system can handle character-level tasks and carry knowledge across low-resource languages.

## 1 Introduction

Language models are about uncovering patterns in a sequence so they can guess what comes next. Before any of that happens, we must decide what the pieces of that sequence—the *tokens*—actually are. That choice is usually frozen in advance by a *tokeniser* that chops raw text into discrete units long before training begins. Consider the sentence "The quick brown fox." A *character*-level tokeniser feeds the model the stream {`T, h, e, ␣, q, u`} and asks it to predict the next letter `i`. A *word*-level tokeniser, in contrast, hands over {`The`, `quick`} and expects the model to guess `brown` in one shot. Finer cuts lead to larger sequences and shorten the look-ahead window, whereas coarser cuts lead to shorter sequences but make each token rarer and harder to compare and predict. Regardless of granularity, some form of tokenisation is unavoidable: a sequence must exist before any Transformer can run.

**Byte-Pair Encoding** (BPE) followed by a simple embedding table is by far the most popular approach. It works by repeatedly merging the most frequent byte sequences in the training text until a preset vocabulary limit is reached. This procedure leaves practitioners with just two intuitive *dials*. The first dial is the *training corpus*: whichever text one feeds the algorithm—English prose, source code, or a multilingual mix—determines which patterns are merged and therefore what the final tokens look like. The second dial is the *vocabulary size*: raising this limit lets the merge process run for more steps, producing longer tokens and shorter sequences at the cost of a larger embedding table and output softmax.

*Equal contribution
Code open-sourced at https://github.com/facebookresearch/lingua/tree/main/apps/aunet

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

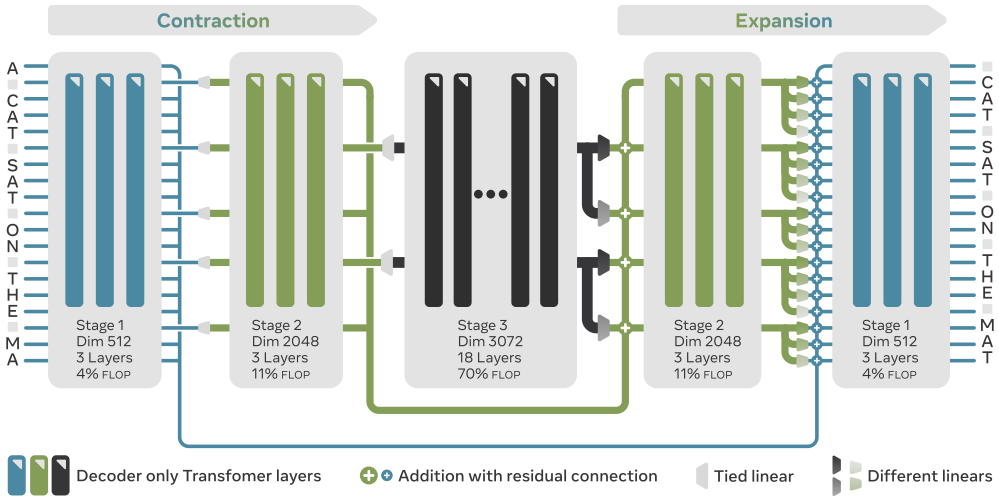


Figure 1: Three-stage Autoregressive U-Net (AU-Net). The model executes from left to right. The contracting path compresses the sequence in two steps: Stage 1 processes raw bytes, Stage 2 keeps only the vector at each word boundary, and Stage 3 keeps one vector per two words. Each contraction and expansion step supports arbitrary pooling and upsampling patterns. After the deepest stage, the expanding path reverses the contracting path by duplicating each coarse vector and applying position-specific linear layers. These are combined with skip connections from the contracting path, gradually restoring sequence length and blending in high-level information. Deeper stages predict further ahead and capture broad semantics, while shallower stages refine local detail.

Most issues with tokenisation stem from the embedding operation rather than the splitting act itself. Each token is typically mapped to an independent vector, meaning the network sees only opaque identifiers and must rediscover, for instance, that *strawberry* and *strawberries* share nine letters. This reliance on isolated embeddings hampers symbol-level reasoning and complicates transfer to dialects or rare languages. Finally, this splitting is most often a preprocessing step, locking in a single level of granularity for all subsequent model layers (see Section 2.2).

To address these limits, our **Autoregressive U-Net** (Section 2.1), or AU-Net ('oh-net', /óʊ nɛt/), learns to embed information directly from raw bytes, and allows for multiple stages of splitting. The purpose of an embedding is to map tokens to vectors. Instead of using a lookup table, we use attention directly to embed the tokens. Self-attention allows vectors at any position to summarize the entire preceding context. This enables a simple pooling mechanism: we select these contextualized vectors at word boundaries (AU-Net-2), then word pairs (AU-Net-3), and up to four-word chunks (AU-Net-4), forming a multi-stage embedding hierarchy. This U-Net like architecture contracts sequences, preserving detail with skip connections, before expanding them. During expansion, vectors representing coarser information are injected back into more fine-grained representations.

Deeper stages operate on compressed representations, allowing them to aggregate information over longer spans of text. While the model remains strictly autoregressive and performs next-byte prediction, the hierarchical structure introduces an inductive bias that encourages the formation of more abstract representations akin to multi-token prediction [1] but achieved without auxiliary losses. This effect allows deeper stages to guide shallower stages at the semantic level, while letting them handle finer details like spelling.

**Contributions** (quantified in Section 3).

**C1.** *Adaptive multi-level hierarchy*. We train up to four end-to-end embedding stages with arbitrary, user-specified split functions, extending prior work that relies either on fixed pooling or shallow hierarchies.

**C2.** *Infinite vocab size*. By operating directly on bytes, our model avoids predefined vocabularies and memory-heavy embedding tables, enabling open-vocabulary modeling without increasing memory footprint. This property is inherent to byte-level approaches, and our results validate its effectiveness at scale.

**C3.** *Strong performance and scaling*. Under identical pre-training budgets, a single level matches strong BPE baselines, and a two or three-level hierarchy shows promising scaling trends. A selection of the results is presented in Table 1

**C4.** *Practical Efficiency*. We maintain comparable GPU throughput in wall-clock time instead of purely theoretical compute gains. Our code is available in Meta Lingua [2].

**C5.** *Stable scaling laws*. We show that moving from token to byte-level training demands new batch size and learning rate formulas to get smooth optimization.

By turning a one-shot, memory-hungry embedding into a learned, multi-scale process, we offer a flexible alternative to the rigid BPE preprocessing followed by a simple embedding table. Table 1 provides a concise overview of results obtained at the 1B scale on 370B tokens, comparing AU-Net-2 (two-stage), AU-Net-3 (three-stage), and AU-Net-4 (four-stage) variants. These results highlight the promising performance in the heavily overtrained regime.

## 2 Method

### 2.1 Autoregressive U-Net

Inspired by U-Net-like architectures [3, 4], we propose an autoregressive hierarchical model for language modelling, illustrated in figure 1. This architecture features a *contracting path*, which compresses the input sequence, and an *expanding path*, which reconstructs it. Both paths are fully *adaptive*: they do not require fixed pooling or upsampling sizes. Pooling and upsampling operations can be designed independently, even if we choose to make them symmetrical in this paper. The only requirement is a *splitting function*, which specifies the positions in the sequence where pooling should occur. This function is detailed in section 2.2.

Table 1: 1B equivalent on 370B tokens

| Model | FLOP | Hellaswag | MMLU | GSM8k |
|---|---|---|---|---|
| BPE | 4e21 | 70.2 | 27.0 | 4.4 |
| AU-Net 2 | 3e21 | 69.9 | 28.8 | 3.0 |
| AU-Net 3 | 4e21 | 72.9 | 28.0 | 3.7 |
| AU-Net 4 | 5e21 | **73.7** | **31.7** | **5.3** |

Our architecture is *monolithic*: unlike recent approaches [5, 6] that use local models, we apply attention globally at each stage (or within a sliding window), allowing every input to attend to previous inputs. This ensures that words or word groups are not processed in isolation. To preserve fine-grained information that might be lost during contraction, we introduce skip connections between stages, following the approach in Ronneberger et al. [3] and Nawrot et al. [4]. We also increase the hidden dimension at each stage in proportion to its contraction factor, enabling richer representations as the sequence is contracted. To keep computation tractable at the byte-level stage (Stage 1), where sequences are longest, we restrict attention to a window.

#### 2.1.1 Pooling and Upsampling

Since our pooling and upsampling are adaptive, we cannot rely on fixed window sizes. To address this, we explored several pooling and upsampling strategies. In this section, we describe the method used in all experiments reported in the main text. A complete description of the alternatives and ablation results can be found in the appendix C.

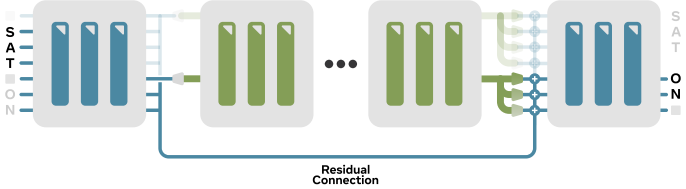


Figure 2: Pooling simply selects the vectors at the positions specified by the splitting function. Upsampling then expands each pooled vector to fill the next segment, applying a separate linear layer for each position. For instance, the pooled vector representing the word 'SAT␣' is used to help predict 'ON␣'. This offset lets deeper stages predict further ahead in the sequence. When using 4 stages, for example, this results in the deepest stage helping for the prediction of the next four words.

**Pooling.** We adopt the simplest pooling strategy: selecting the indices identified by the splitting function and projecting them to the next stage's dimensionality using a linear layer. Since the preceding layers already include attention mechanisms, we rely on these to do the pooling implicitly instead of relying on explicit cross attention as used in Nawrot et al. [4], Pagnoni et al. [5].
Formally, let $X \in \mathbb{R}^{s \times d_{\text{in}}}$ denote the sequence of hidden states, and let $I = \{i_1 < \cdots < i_m\} \subseteq \{1, \ldots, s\}$ be the indices selected by the splitting function, defining the new sequence length $m$ with $1 < m \leq s$. Let $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ be a learnable projection, where $d_{\text{in}}$ and $d_{\text{out}}$ correspond to the dimensions of the current and following stages, respectively. The pooled (downsampled) sequence is obtained by selecting the indexed rows and applying the linear projection:

$$Y = X_I W \in \mathbb{R}^{m \times d_{\text{out}}},$$

where $X_I = [X_{i_1}; \ldots; X_{i_m}]$.

**Upsampling.** The upsampling step maps coarse representations to finer ones for the next stage. As illustrated in Figure 2, we duplicate each coarse vector to match the length of the **following** segment, applying distinct, position-specific linear transformations to these duplicates.
Let $Y \in \mathbb{R}^{m \times d_{\text{out}}}$ and position-specific projections $\{W_p\}_{p=1}^{K}$ with $W_p \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Given per-segment lengths $r_i \in \{1, \ldots, K\}$, compute

$$X_{(i,p)} = Y_i W_p \quad \text{for } i = 1, \ldots, m, p = 1, \ldots, r_i.$$

Stacking $(i, p)$ in segment-major order gives $X \in \mathbb{R}^{(\sum_{i=1}^{m} r_i) \times d_{\text{in}}}$.
Since these transformations are shared across segments but vary by position within a segment, we term this *Multi-Linear Upsampling*. In our experiments, models with multiple stages are more sensitive to the specific choice of upsampling strategy, whereas for pooling, many strategies work equally well.

### 2.1.2 Generation

During training, we process the entire input sequence in parallel, activating all stages simultaneously. At inference, generation is autoregressive: the byte-level stage is active at every step, while deeper stages activate less frequently according to the pooling pattern. Skip connections transmit information upward at each stage, so deeper stages can integrate fine-grained details. This cascading, conditional activation enables efficient inference: computationally intensive high-level stages activate rarely, but still effectively guide detailed lower-level predictions. In practice, this means that we need to cache the latest vector at the output of each stage to correctly propagate deeper stages' outputs.

## 2.2 Splitting Function

The AU-Net architecture supports flexible splitting strategies to define pooling points at each hierarchical stage. The primary constraint is that any chosen splitting function must be *stable to rightward insertion*: appending bytes should not alter prior pooling decisions, ensuring consistent autoregressive generation. Various methods (e.g., fixed windows [4], entropy [5], learned rules) are possible. Our current work splits on spaces using different regular expressions at each stage (details in Appendix B).

This strategy defines a hierarchy: Stage 1 processes raw bytes; Stage 2 pools at word boundaries (identified by the regex); Stage 3 pools after every two words (or sentence end); and Stage 4 after every four words (or sentence end). This rule-based approach, inspired by pre-tokenization in systems like GPT-4o's [7], is effective for Latin scripts. Extending robustly to languages without clear delimiters remains future work. Unlike prior approaches [5, 6, 8] that used similar splits mainly to replace BPE in a single-stage context, AU-Net uses these user-defined splits for its multi-stage hierarchical processing.

## 2.3 Evaluating on different scales

Large language models scale very predictably [9–11]. This allows us to estimate the performance of a model for a large compute budget. But more surprisingly, it allows us to predict the optimal hyperparameters for models way beyond our ablation budget. Bi et al. [11] described a method for sweeping learning rates (LR) and batch sizes (BSZ) across a range of small models, and they demonstrated that these results can be used to predict optimal hyperparameters for larger models.

Following their methodology, we show a different evolution of hyperparameters, both due to the data in our setup and to the hierarchical model. These hyperparameters are then used to do scaling laws for a bigger range of compute budgets to compare the baseline architecture and AU-Net. Throughout this paper, the *scale* of a run is its total pre-training compute $C$ measured in Floating Point Operation (FLOP):

$$C = \underbrace{F_{\text{model / input-unit}}}_{\text{FLOPs per (forward+backward) pass per input unit}} \times \underbrace{N_{\text{input-unit}}}_{\text{number of units of training input}}.$$

Following Bi et al. [11], we define model size as the number of FLOPs per input unit instead of relying on the number of parameters. This allows us to compare models with different architectures fairly. The formula for the number of FLOP per input-unit for a decoder-only transformer is given by:

$$F_{\text{model / input-unit}} = \underbrace{6N_{\text{params}}^{\text{no-embed}}}_{\text{linear term}} + \underbrace{6d\,L\,S}_{\text{attention term}}.$$

where, $N_{\text{params}}^{\text{no-embed}}$ is the number of parameters, excluding the embeddings. $d$ is the dimension, $S$ the sequence length and $L$ the number of layers. To scale up, one can either make the model bigger ($F_{\text{model / input-unit}} \uparrow$), give it more data ($N_{\text{input-unit}} \uparrow$), or do both. Gadre et al. [12] showed that keeping the *data-to-model ratio* $\gamma_{\text{input-unit}}$ constant is key to getting smooth scaling laws and predictable performance, where:

$$\gamma_{\text{input-unit}} = \frac{N_{\text{input-unit}}}{F_{\text{model / input-unit}}}.$$

We adopt this convention in all experiments and report the data-to-model ratio $\gamma_{\text{input-unit}}$ used in the experiments.

**Bytes versus tokens.** On DCLM [13], a token sequence is on average $k \approx 4.56$ times shorter than its byte sequence when using the LLaMa 3 tokenizer.

Given some compression factor $k$ between bytes and tokens, we want to express the equivalent $\gamma_{\text{bytes}}$. To do this, we note that $N_{\text{byte}} = k \times N_{\text{token}}$ and $F_{\text{model/byte}} = F_{\text{model/token}}/k$. Therefore,

$$\gamma_{\text{byte}} = k^2 \frac{N_{\text{token}}}{F_{\text{model/token}}} = k^2 \gamma_{\text{token}}.$$

Note that this scaling relationship is architecture-agnostic. The factor $\frac{1}{k}$ follows directly from the difference in sequence length between tokens and bytes under a given tokenizer. While different architectures may have distinct FLOPs/byte, the conversion between token- and byte-level compute is determined solely by the compression ratio $k$. This factor allows us to compare the performance of our model with the baseline on the same scale, as they will have seen the same amount of data and spent the same amount of FLOPs per token. Throughout the paper, we always express the data-to-model ratio in LLaMa 3 tokens ($\gamma_{\text{token}}$).

**FLOPS per byte for AU-Net.** In the case of AU-Net, we cannot use the same formula as the baseline because of the contraction and expansion happening in the model. However, we can still use the same formulas as long as we account for the contraction at each stage. So the total FLOPs per byte for AU-Net is simply the sum of each stage divided by the contraction factor.

$$F_{\text{model/byte}} = \sum_{i=1}^{L} \frac{F_{\text{model/byte}}^{i}}{k_i},$$

where $k_i$ is the contraction factor at stage $i$.

This property allows us to have models with a higher number of parameters for the same compute budget and data-to-model ratio.

**Hyperparameter scaling laws** Bi et al. [11] showed that the regularity of scaling laws can be exploited to tune very large models from a sweep over much smaller ones. We replicate their protocol on six miniature versions of each architecture (baseline Transformer and AU-Net): we perform a quasi-random search over batch size and learning rate, keep the configurations within 1% of the best validation loss, and fit $\text{BSZ}(C) = A\,C^{\alpha}$ and $\text{LR}(C) = B\,C^{\beta}$ to those points, with parameters $A, \alpha, B$ and $\beta$. We find the following formulas at the byte level for AU-Net:

$$\text{BSZ}_{\text{AU-Net}}(C) = 0.66 \times C^{0.321} \qquad \text{LR}_{\text{AU-Net}}(C) = 6.6 \times C^{-0.176}.$$

And we run the same tuning for the BPE baseline, for which we find:

$$\text{BSZ}_{\text{BPE}}(C) = 29.9 \times C^{0.231} \qquad \text{LR}_{\text{BPE}}(C) = 19.3 \times C^{-0.177}.$$

# 3 Experimental Results

## 3.1 Experimental Setup

**Data.** For all experiments, DCLM [13] served as the pretraining dataset, with a small portion held out for validation, totalling around 4T tokens (of GPTNeoXTokenizer). The corpus is mostly English and focuses on natural language understanding, i.e. it contains a marginal amount of code and maths.
**Baselines.** We compare our approach to different three baselines: Transformers equipped with the BPE tokenizer of LLaMa 3, Transformers and Mamba [14] trained directly on bytes, AU-Net variants using a fixed-size pooling window (denoted Transformer bytes[$::w$], where $w$ is the window size), similar to the Hourglass architecture [4]. To keep the comparison fair, we trained each baseline with the same amount of data or compute. For example, if a data budget of 273B bytes is used to train the bytes level or AU-Net model, this budget is converted to 60B training tokens for a transformer with LLaMa 3 tokenizer [15] because of the $4.56$ compression rate measured on the DCLM corpus.
**AU-Net Architecture.** AU-Net-2, -3, and -4 progressively increase embedding dimensionality across stages ($512 \rightarrow 2048 \rightarrow 3072 \rightarrow 4608$ for deeper variants at the 1B scale), with layer allocation guided by the ablation results in appendix C.2. The AU-Net-2 architecture at 1B scale is illustrated in figure 1, showing stage dimensions, layer distribution, and total FLOP allocation per stage. The first byte-level stage does not require high dimensionality, as it primarily encodes local information before compression, whereas later stages capture increasingly abstract representations that benefit from greater capacity. Contraction rates are chosen so that when representations are merged, information density remains approximately constant (e.g., a sequence compressed by a factor of two doubles its hidden dimension). At the 1B scale, AU-Net-4 uses $6 + 25$ layers with a maximum hidden size of 4608, while the 8B-scale configuration expands to $6 + 33$ layers and a final hidden size of 4096. This setup reflects the model's compression rationale while keeping compute distribution flexible. For comparison, the BPE Transformer employs 25 layers with a hidden size of 2048 at the 1B scale and 33 layers with a hidden size of 4096 at the 8B scale.

Full architectural specifications, such as embedding sizes, layer counts, are detailed in appendix C.2. Also, note that, due to compute constraints, AU-Net-3 and AU-Net-4 were not trained at the 8B scale.
**Hyperparameters.** For a detailed overview of the hyperparameters, see appendix D. As explained in section 2.3, we sweep batch size and learning rate values across model scales ranging from 25M to 500M. Then, we extrapolate the best learning rate and batch size for any given compute budget.
**Evaluation Metrics.** All models are evaluated on a broad set of downstream tasks in a zero-shot setting, occasionally including a few in-context examples directly in the prompt. These tasks fall into two categories: (i) multiple-choice (MCQ) tasks, where the correct answer is selected as the option with the lowest normalized negative log-likelihood (divided by the number of characters) [16]; and (ii) open-ended generation tasks, where the model is allowed to freely generate its answer.

To highlight the strengths of AU-Net, we include specialized benchmarks targeting character-level manipulation (CUTE [17] appendix E) and low-resource language translation (FLORES-200, [18] section 3.4).

For clarity, we report a selection of key benchmark results in the main tables, including Hellaswag, ARC-Easy, ARC-Challenge, MMLU, NQ, TQA, and GSM8K. Also, we report 95% confidence intervals for all tables using bootstrap. A full breakdown of all evaluation results is provided in the appendix F.

In addition to task performance, the total training FLOPs and training throughput are provided for each model, measured in bytes per second per GPU (bps) on H100 80GB GPUs (internal cluster) during the actual training.
**Implementation Details.** As scaling is key to the success of large language models, our implementation balances efficiency and simplicity. We use *sequence packing* along with full attention, a strategy shown to have little to no impact on downstream performance ([13]). To reduce GPU memory pressure, all our experiments rely on Fully Sharded Data Parallelism (FSDP).

For additional speed-ups, the entire model is compiled with `torch.compile`. Compilation, however, requires a static computation graph, which clashes with the variable-length outputs produced by our adaptive pooling: the number of bytes per word (and thus per stage) naturally varies across sentences. We resolve this by fixing a maximum sequence length at every stage: sequences that exceed the limit are truncated abruptly, and shorter ones are padded. This compromise yields a graph that is static for compilation while still supporting adaptive hierarchical pooling in practice.

Table 2: **Downstream results comparing AU-Net to BPE and byte-level baselines.** We report accuracy on key benchmarks with 95% confidence intervals where applicable. Literature models are shown in *italics*; all models are trained on the same corpus, unless specified. AU-Net variants differ in the number of stages. We also report compute budget and empirical training speeds in bytes/sec.

| Model | Params | Emb. | Flops | bps | Hellaswag | ARC_E | ARC_C | MMLU | NQ | TQA | GSM8k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dim=2048 (1B model), 60B tokens (data-to-model ratio of 10)** | | | | | | | | | | | |
| Mamba Byte | 1.3B | 1M | 3e21 | 32k | 63.0 ±0.9 | 60.3 ±2.0 | 33.6 ±2.8 | 25.1 ±0.7 | 8.2 ±0.9 | 21.2 ±0.7 | 2.1 ±0.8 |
| Transformer Byte | 1.3B | 1M | 4e21 | 47k | 63.0 ±1.0 | 61.2 ±1.9 | 34.7 ±2.7 | 24.7 ±0.7 | 8.8 ±0.9 | 21.4 ±0.8 | 2.5 ±0.9 |
| Transformer Byte[::4] | 1.3B | 1M | 6e20 | 180k | 63.5 ±0.9 | 63.0 ±1.9 | 36.0 ±2.7 | 25.1 ±0.7 | 6.8 ±0.8 | 17.2 ±0.7 | 2.6 ±0.9 |
| Transformer Byte[::5] | 1.3B | 1M | 5e20 | 218k | 60.0 ±1.0 | 60.0 ±2.0 | 34.8 ±2.8 | 23.9 ±0.7 | 5.6 ±0.7 | 14.7 ±0.7 | 2.0 ±0.8 |
| Transformer Byte[::6] | 1.3B | 1M | 4e20 | 255k | 58.6 ±1.0 | 59.5 ±2.0 | 32.4 ±2.7 | 24.9 ±0.7 | 4.6 ±0.7 | 12.3 ±0.6 | 2.3 ±0.8 |
| AU-Net 2 | 1.3B | 1M | 5e20 | 225k | 64.2 ±0.9 | 64.4 ±1.9 | 35.2 ±2.8 | 24.8 ±0.7 | 8.8 ±0.9 | 20.4 ±0.7 | 2.7 ±0.9 |
| AU-Net 3 | 2.5B | 1M | 7e20 | 180k | **67.4** ±0.9 | 65.9 ±1.9 | 36.7 ±2.7 | **26.3** ±0.7 | **9.6** ±1.0 | 22.6 ±0.8 | 2.3 ±0.8 |
| AU-Net 4 | 4.2B | 1M | 8e20 | 155k | 66.4 ±0.9 | **67.4** ±1.9 | **37.0** ±2.8 | **26.3** ±0.7 | 5.1 ±0.7 | 15.5 ±0.7 | **3.5** ±1.0 |
| Transformer BPE | 1.8B | 525M | 7e20 | 210k | 63.6 ±1. | 62.8 ±1. | 36.5 ±2. | 26.2 ±0. | 8.8 ±0.9 | **26.3** ±0. | 2.3 ±0.8 |
| **Dim=2048 (1B model), 370B tokens (data-to-model ratio of 40)** | | | | | | | | | | | |
| AU-Net 2 | 1.3B | 1M | 3e21 | 225k | 69.9 ±0.9 | 68.6 ±1.9 | 38.9 ±2.7 | 28.8 ±0.7 | 13.0 ±1.1 | 32.5 ±0.9 | 3.0 ±0.9 |
| AU-Net 3 | 2.5B | 1M | 4e21 | 180k | 72.9 ±0.9 | 72.3 ±1.8 | **43.3** ±2.8 | 28.0 ±0.7 | **15.3** ±1.2 | **39.1** ±0.9 | 3.7 ±1.0 |
| AU-Net 4 | 4.2B | 1M | 5e21 | 155k | **73.7** ±0.9 | **72.6** ±1.8 | 43.2 ±2.9 | **31.7** ±0.7 | 14.0 ±1.1 | 35.5 ±0.9 | **5.3** ±1.2 |
| Transformer BPE | 1.8B | 525M | 4e21 | 210k | 70.2 ±0. | 68.6 ±1. | 38.5 ±2. | 27.0 ±0. | 13.5 ±1. | 37.2 ±0. | 4.4 ±1.1 |
| *DCLM-1B-5×(145B)*[13] | 1B | 207M | 1e21 | - | 66.1 | 70.2 | 40.6 | 26.4 | - | 29.3 | 1.1 |
| *MegaByte (263B)*[19] | 1.1B | - | - | 73k | 38.9 | 54.9 | 23.4 | 25.1 | - | 9.6 | |
| *Hierarchical (263B)*[6] | 1.1B | - | 1e21 | - | 46.5 | 65.0 | 30.5 | 26.0 | - | 9.6 | - |
| **Dim=4096 (8B model), 200B tokens (data-to-model ratio of 5)** | | | | | | | | | | | |
| AU-Net 2 | 7.9B | 1M | 1e22 | 41k | **79.1** ±0.8 | **80.0** ±1.6 | **51.2** ±2.9 | **51.1** ±0.8 | **22.1** ±1.3 | **50.9** ±0.9 | **10.0** ±1.6 |
| Transformer BPE | 7.5B | 1B | 9e21 | 43k | 77.2 ±0. | 74.5 ±1. | **49.2** ±2. | 49.6 ±0. | **21.1** ±1. | **51.1** ±0. | **10.7** ±1.7 |
| *DCLM-7B-2×(276B)*[13] | 7B | 413M | 1e22 | - | 77.8 | 78.1 | **52.6** | **50.8** | - | **50.9** | 4.3 |
| *Hierarchical (263B)*[6] | 9.2B | - | 1e22 | 15k | 56.3 | 76.6 | 44.2 | 32.0 | - | 33.1 | - |
| *BLT (220B)*$^{\times}$[5] | 8B | - | 1e22 | - | 72.2 | 66.8 | 38.8 | 25.2 | - | - | - |
| *BLT (1T)*$^{*}$[5] | 8B | - | 5e22 | - | 80.6 | 79.6 | 52.1 | 57.4 | - | - | - |
| *DCLM-7B (2.5T)*$^{*}$[15] | 7B | 413M | 1e23 | - | 80.4 | 82.2 | 59.9 | 63.7 | - | 52.7 | 2.5 |
| *LLaMa 3.1 (15T)*$^{\times}$[15] | 8B | 1B | 6e23 | - | 83.3 ±0.8 | 80.7 ±1.5 | 54.8 ±2.9 | 66.4 ±0.8 | 29.1 ±1.5 | 64.4 ±0.9 | 54.7 ±2.7 |

$^{*}$ Trained on mix of DCLM and other datasets

$^{\times}$ Trained on different corpus than DCLM

### 3.2 Equal Data Budget Results

We evaluate the effectiveness of hierarchical pooling by fixing the model's primary hidden dimension to 2048 and maintaining a constant total training-data budget. The hidden dimension at each stage is scaled proportionally to its contraction ratio as described in section 2.1. For instance, the byte-level stage uses a dimension of $2048/4 = 512$, the word-level stage uses 2048, and the 2-word level uses $1.5 \times 2048 = 3072$, continuing in this manner for deeper stages. We assess the downstream performance of language models with 2, 3, and 4 stages at the 1B parameter scale. For the 8B model, we evaluate only the 1-stage configuration for now. All variants are compared against a Transformer baseline using the LLaMA 3 tokenizer of the same main hidden dimension. More ablations regarding pooling and the number of layers per stage can be found in the appendix C.

As shown in table 2, hierarchical models consistently match or outperform their BPE-based counterparts. This trend holds across various configurations and becomes especially pronounced as we introduce more hierarchical stages. Notably, multi-stage AU-Net models (e.g., AU-Net 3 and AU-Net 4) outperform BPE baselines on several benchmarks. An interesting exception to this pattern is the TQA benchmark, which is a knowledge-intensive task evaluating the generation of the model. AU-Net models along with byte-level baselines consistently underperform on TQA compared to BPE-based models. This suggests that the performance gap may not stem solely from the hierarchical structure. However, as model size and training data scale (e.g., at the 8B or 1B, 370B tokens scale), this discrepancy seems to vanish. When examining the Transformer Bytes[::$w$] models, we clearly observe the effect of pooling at word boundaries. As the pooling window size $w$ increases, the resulting increasing compression comes at the cost of a consistent drop in performance. In contrast, using a more principled pooling strategy, as in AU-Net, achieves better compression while main-

taining strong downstream performance, enabling more robust and faster training. This effect is particularly pronounced on generative tasks such as TQA, where performance declines sharply for larger window sizes and remains below both AU-Net and BPE baselines. This is further emphasised in the subsequent section 3.4 where the same pattern emerges. For languages where space-based pooling does not align well with word boundaries, performance also decreases significantly.

We observe early signs of diminishing returns beyond a certain number of stages. While AU-Net 4 improves on reasoning-heavy tasks such as ARC-C and GSM8k, gains on benchmarks like Hellaswag and TQA are less consistent. However, this effect may stem not from hierarchy itself, but from data efficiency: deeper hierarchies might require more training data to reach their full potential. Supporting this interpretation, AU-Net 2 and AU-Net 4 benefit significantly from additional training data, and that MMLU and GSM8k scores continue to improve with more stage, even at fixed scale.

Finally, when comparing our models to similarly sized baselines from the literature (italicized in the table), we find that AU-Net remains competitive, even while using significantly less training data. For instance, BLT (1T) uses approximately 5× more compute than our 8B model, while only being better on MMLU. Importantly, comparisons with literature models are fair, as all were trained on the same corpus: DCLM (except for BLT (220B) and LLAMA 3.1 (15T)).

To further evaluate our approach, we now turn to scaling laws (figure 3) to better quantify how our architecture compares to a standard Transformer with BPE. We focus on AU-Net 2 and AU-Net 3, using a data-to-model ratio of 2. This choice is motivated by the diminishing returns observed when moving from AU-Net 3 to AU-Net 4 under the same data-to-model ratio.

### 3.3 Scaling laws

Using the learning rate and batch size formulas (Section 2.3), we run pretraining for a range of compute budgets ranging from 1e19 to 1e22 flops (corresponding to models from 150M to 5.3B non embedding parameters) for the baseline, with a data-to-model ratio of 10. This is roughly 2× the optimal data-to-model ratio found by Kaplan et al. [9]. Compared to table 2 and the 8B-scale experiments, the scaling-law runs use a higher data-to-model ratio (data-to-model ratio = 10 instead

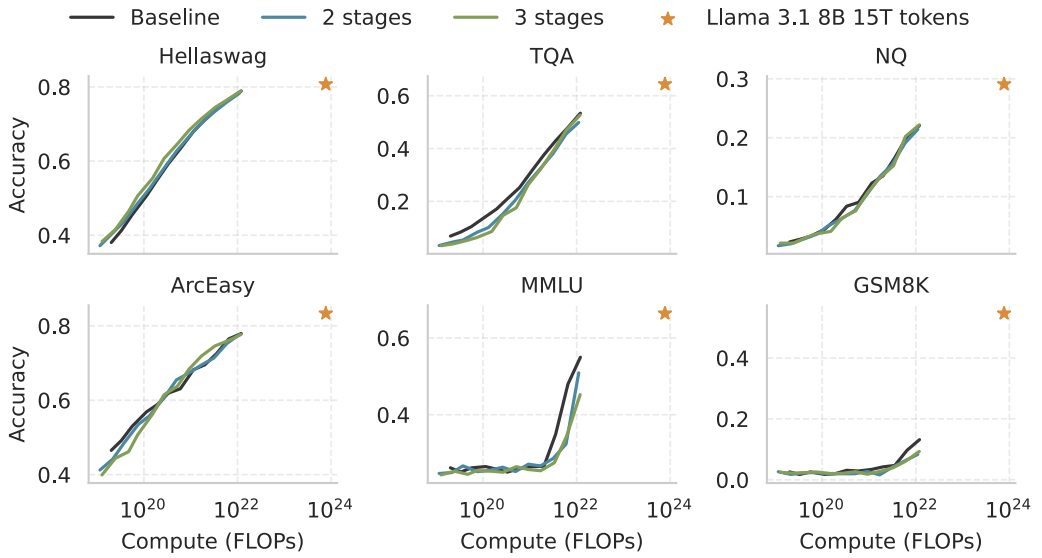


Figure 3: Downstream task performance scaling with compute (1e19-1e22 FLOPs) under data-to-model ratio of 10. AU-Net (2/3 stages) generally tracks a strong BPE Transformer baseline, which itself performs competitively against much larger models (e.g., LLaMa 3.1 8B on 15T tokens 100× compute). While AU-Net matches the baseline on tasks like Hellaswag and ARC Easy, and catches up on TQA at higher compute, its performance improvement phase on MMLU and GSM8K appears to start later. The general underperformance on GSM8K is also linked to limited math data in the DCLM pretraining corpus.

Table 3: Multilingual evaluation. **Left:** BLEU scores on the FLORES-200 benchmark across multiple languages. Higher scores indicate better translation quality. **Right:** MMLU Exact Match (%) across 26 non-English languages. Results are averaged per language across all tasks.

| **FLORES-200** (BLEU) | Lang. → Eng. BPE | Lang. → Eng. AU-Net 2 | Eng. → Lang. BPE | Eng. → Lang. AU-Net 2 |
|---|---|---|---|---|
| German | **34.4** ±1.2 | 33.9 ±1.2 | **16.7** ±0.8 | 15.6 ±0.7 |
| Dutch | 24.7 ±1.0 | **25.0** ±1.0 | **12.3** ±0.6 | 11.7 ±0.6 |
| Afrikaans | 32.0 ±1.3 | **35.7** ±1.3 | 14.8 ±0.8 | **16.1** ±0.8 |
| Faroese | 8.7 ±0.7 | **9.9** ±0.8 | 1.8 ±0.3 | **2.9** ±0.4 |
| Icelandic | 7.8 ±0.6 | **9.0** ±0.7 | 1.7 ±0.3 | **2.5** ±0.3 |
| Limburgish | 15.3 ±0.9 | **19.9** ±1.0 | 5.7 ±0.4 | **6.7** ±0.5 |
| Luxembourgish | 11.4 ±0.8 | **14.7** ±0.9 | 2.6 ±0.3 | **4.0** ±0.3 |
| Italian | 29.1 ±1.0 | **30.1** ±1.0 | 15.1 ±0.7 | **15.3** ±0.6 |
| Friulian | 14.6 ±0.8 | **19.1** ±1.0 | 3.2 ±0.3 | **4.0** ±0.3 |
| Ligurian | 16.5 ±0.9 | **21.8** ±1.0 | 3.4 ±0.3 | **3.9** ±0.3 |
| Lombard | 12.9 ±0.9 | **19.2** ±1.0 | **5.2** ±0.4 | 4.2 ±0.3 |
| Sardinian | 14.3 ±0.8 | **18.2** ±1.0 | 4.3 ±0.4 | **4.5** ±0.4 |
| Sicilian | 11.7 ±0.8 | **16.8** ±0.9 | 3.9 ±0.4 | **4.7** ±0.4 |
| Venetian | 19.8 ±1.0 | **25.4** ±1.1 | **5.8** ±0.4 | 5.6 ±0.4 |
| Spanish | 28.2 ±1.0 | **29.3** ±1.0 | **20.2** ±0.7 | 19.8 ±0.7 |
| Asturian | 24.0 ±1.1 | **28.6** ±1.1 | **10.3** ±0.6 | 8.2 ±0.5 |
| Catalan | 28.1 ±1.1 | **33.0** ±1.2 | 9.6 ±0.5 | **10.7** ±0.6 |
| Occitan | 28.0 ±1.2 | **35.5** ±1.2 | 4.8 ±0.4 | **6.2** ±0.4 |
| Portuguese | 42.0 ±1.3 | **43.6** ±1.3 | 25.3 ±1.0 | **25.4** ±1.0 |
| Galician | 29.6 ±1.1 | **34.0** ±1.2 | 9.9 ±0.5 | **10.2** ±0.6 |
| Papiamento | 17.3 ±0.9 | **22.1** ±1.1 | 2.5 ±0.3 | **6.3** ±0.4 |
| Kabuverdianu | 13.7 ±0.9 | **20.8** ±1.1 | 2.4 ±0.3 | **5.1** ±0.4 |
| Esperanto | 15.9 ±1.0 | **19.3** ±1.0 | 3.6 ±0.4 | **5.9** ±0.4 |
| **Average** | 20.9 ±0.2 | **24.6** ±0.2 | 8.0 ±0.1 | **8.7** ±0.1 |

| **MMLU** | BPE | AU-Net 2 |
|---|---|---|
| English | 49.6 ±0.8 | **51.1** ±0.8 |
| Arabic | 29.1 ±0.8 | **29.5** ±0.8 |
| Bengali | 27.5 ±0.7 | **27.6** ±0.8 |
| Chinese | **33.0** ±0.8 | 28.0 ±0.7 |
| Czech | 30.7 ±0.8 | **32.2** ±0.8 |
| Dutch | 34.5 ±0.8 | **37.1** ±0.8 |
| Finnish | 29.0 ±0.7 | **29.3** ±0.7 |
| French | 37.3 ±0.8 | **40.7** ±0.8 |
| German | 36.0 ±0.8 | **37.6** ±0.8 |
| Greek | 29.2 ±0.8 | **30.5** ±0.8 |
| Hindi | **27.9** ±0.7 | 27.5 ±0.7 |
| Hungarian | 29.0 ±0.8 | **30.1** ±0.8 |
| Indonesian | 34.9 ±0.8 | **37.3** ±0.8 |
| Italian | 36.2 ±0.8 | **39.0** ±0.8 |
| Japanese | **29.5** ±0.7 | 28.2 ±0.7 |
| Korean | **28.4** ±0.7 | 28.2 ±0.8 |
| Persian | **28.7** ±0.7 | 28.6 ±0.7 |
| Polish | 30.3 ±0.8 | **32.0** ±0.8 |
| Portuguese | 37.2 ±0.8 | **40.9** ±0.8 |
| Romanian | 34.0 ±0.8 | **36.9** ±0.8 |
| Russian | 30.9 ±0.8 | **31.2** ±0.8 |
| Spanish | 37.6 ±0.8 | **41.4** ±0.8 |
| Swahili | 28.8 ±0.7 | **29.9** ±0.8 |
| Swedish | 33.5 ±0.8 | **36.0** ±0.8 |
| Telugu | 26.8 ±0.7 | **27.4** ±0.7 |
| Thai | **28.0** ±0.7 | 27.5 ±0.7 |
| Turkish | 29.1 ±0.7 | **30.0** ±0.7 |
| Vietnamese | **31.4** ±0.8 | 30.7 ±0.7 |
| **Average** | 31.4 ±0.1 | **32.4** ±0.1 |

of 5) and the latest hyperparameters described at the end of section 2.3. These differences can explain the variation in peak performance across the two experiments.

The list of models chosen for each budget is detailed in the appendix G. Figure 3 shows the evolution of performance on 6 downstream tasks for AU-Net and the BPE baseline. Here we mainly notice that 2 and 3 stage AU-Net models can match the performance of the BPE baseline when carefully controlling for compute budget. This is the case for Hellaswag, Arc Easy, and NQ. For TQA, AU-Net both for 2 and 3 stages starts with a performance gap, but the 3 stage model catches up at 1e22 flops. However, both 2-stage and 3-stage AU-Net models are still behind the BPE baseline at 1e22 flops for GSM8K and MMLU. Most downstream tasks follow a sigmoid pattern: performance is near chance at low compute, then rapidly improves before plateauing. For AU-Net models, this transition appears to occur slightly later on tasks like GSM8K and MMLU, suggesting that the benefits of a deep hierarchy may become more pronounced at larger scales. Nevertheless, on many benchmarks, both our AU-Net variants and our BPE baseline achieve results remarkably close to those of considerably larger models like LLaMa 3.1 8B (pretrained on 15T tokens, representing 100 times more compute than our largest run shown here). This proximity underscores the strength of our BPE baseline, making AU-Net's ability to match or trend towards it particularly noteworthy. The primary exception where this close tracking is less apparent is GSM8K; however, this underperformance across all our models is likely due to the pretraining corpus, as DCLM contains very little math data.

### 3.4 Extended Evaluations

We present results highlighting two specific advantages of byte-level training with AU-Net over BPE-based Transformers: improved performance on multilingual benchmarks (Table 3) and character-level manipulation tasks (Table 7 in the appendix E). Table 3 show surprisingly strong non-English performance of both model, despite DCLM being heavily filtered toward English.

**Cross-lingual generalization within language families.** On the multilingual MMLU benchmark (Table 3 right), languages using Latin scripts consistently benefit from byte-level modeling. We observe strong positive transfer between related languages. Concretely, Germanic languages (German, Swedish, Dutch, etc.) show an average gain of +3.0 points, while Romance languages (Italian, Spanish,

Portuguese, French, etc.) improve by +4.0 points. These results suggest that operating at the byte level helps the model to capture shared orthographic and morphological patterns across related languages.

**Transfer to low-resource languages.** The FLORES-200 benchmark (Table 3 left) includes many low-resource languages that are underrepresented or absent in the training data. This setting allows us to test the model's ability to generalize based on subword morphology and shared linguistic roots. Byte-level modeling provides the flexibility to construct meaningful representations without requiring the presence of these languages in the tokenizer or training corpus. We observe consistent gains in translation tasks into English, where the model must primarily understand the source language. The advantage is particularly clear for languages that share syntactic or morphological traits with more dominant relatives in the same family. This also highlights the robustness of our model: it can produce meaningful translations even with out-of-vocabulary words or forms unseen during training. In the reverse direction (English to low-resource), generation remains more challenging.

## 4 Related Work

Traditional tokenization methods are important for computational efficiency [20–23], but impose fixed granularities. Early attempts to overcome this rigidity explored adaptive vocabularies [24], n-gram combinations [25], or alternative splitting criteria like entropy [5]. Our work, AU-Net, advances this by integrating tokenization and representation learning into a multi-level, autoregressive U-Net architecture that operates directly on bytes.

This hierarchical, adaptive-pooling design distinguishes AU-Net from prior works. For instance, Megabytes [19] introduce a two stage LLM using local models but with fixed-size token blocks, unlike AU-Net's input-adaptive pooling. Neitemeier et al. [6], Byte Latent Transformers (BLT) [5], Dynamic Pooling Transformer (DPT) [26] and SpaceByte [8] also process bytes or use specialized splitting functions. However, they typically aim to replace BPE for a single effective processing stage or use local attention mechanisms. In contrast, AU-Net leverages user-defined splits within a multi-stage architecture featuring distinct pooling strategies that differ from the cross-attention methods in Nawrot et al. [4], Pagnoni et al. [5]. Nawrot et al. [4] defined a similar U-Net architecture but with fixed pooling, much smaller models, and their evaluations mainly focus on perplexity.

Concurrent to our work, H-Net [27] introduces a hierarchical sequence model based on learnable, data-dependent dynamic chunking. Like AU-Net, H-Net operates directly on bytes and constructs up to three stages network. While both models share similar high-level goals, they differ in key mechanisms: H-Net employs a learned dynamic split function, whereas AU-Net relies on predefined, rule-based splitting functions for hierarchical pooling and upsampling.

## 5 Conclusion

This paper introduces AU-Net, an autoregressive U-Net that processes raw bytes and learns hierarchical token representations. By dynamically pooling bytes into words and multi-word chunks, AU-Net eliminates the need for predefined vocabularies and large embedding tables while preserving BPE performance with higher compression. Experiments show that AU-Net matches strong BPE baselines under controlled compute budgets, with deeper hierarchies demonstrating promising scaling trends. Furthermore, its byte-level operation leads to improved performance on character-level tasks and better generalization to low-resource languages. This approach offers a flexible and efficient alternative to traditional tokenization methods, paving the way for more adaptable and versatile language models.

### Limitations and further work

Our work relies on DCLM, an English-only corpus, and currently supports only space-delimited languages with a predefined splitting function. For example, this affects, Chinese, where MMLU scores are lower than the BPE baseline. Another limitation is the lack of evaluation on code and math benchmarks. Since the DCLM corpus contains very little code abd mathematical data, models trained on it perform poorly on these tasks, yielding low and noisy results close to random performance. One extension could be to learn directly the splitting function. On the software side, as the number of parameters increases with the number of stages, FSDP already struggles to overlap computation and communication even at 3/4 stages, it needs a minimum amount of inputs to be fully overlapped.

## References


[1] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *41st International Conference on Machine Learning*, volume 235, pages 15706–15734, 2024.

[2] Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL github.com/facebookresearch/lingua.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[4] Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Findings of the Association for Computational Linguistics*, pages 1559–1571, 2022.

[5] Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, et al. Byte latent transformer: Patches scale better than tokens. *arXiv:2412.09871*, 2024.

[6] Pit Neitemeier, Björn Deiseroth, Constantin Eichenberg, and Lukas Balles. Hierarchical autoregressive transformers: Combining byte- and word-level processing for robust, adaptable language models. In *13th International Conference on Learning Representations*, 2025.

[7] Gautier Dagan, Gabriel Synnaeve, and Baptiste Roziere. Getting the most out of your tokenizer for pre-training and domain adaptation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *41st International Conference on Machine Learning*, volume 235, pages 9784–9805, 2024.

[8] Kevin Slagle. SpaceByte: Towards deleting tokenization from large language modeling. 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 124925–124950, 2024.

[9] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

[10] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *36th International Conference on Neural Information Processing Systems*, 2022.

[11] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv:2401.02954*, 2024.

[12] Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. *arXiv:2403.08540*, 2024.

[13] Jeffrey Li, Alex Fang, Georgios Smyrnis, et al. DataComp-LM: In search of the next generation of training sets for language 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 14200–14282, 2024.

[14] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024.

[15] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

[16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[17] Lukas Edman, Helmut Schmid, and Alexander Fraser. CUTE: Measuring LLMs' understanding of their tokens. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Conference on Empirical Methods in Natural Language Processing*, pages 3017–3026, 2024.

[18] Marta Costa-jussa, James Cross, Onur Çelebi, Maha Elbayad, et al. Scaling neural machine translation to 200 languages. *Nature*, 630, 06 2024.

[19] Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 78808–78823, 2023.

[20] Mehdi Ali, Michael Fromm, Klaudia Thellmann, et al. Tokenizer choice for LLM training: Negligible or crucial? In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics*, pages 3907–3924, 2024.

[21] Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. An analysis of tokenization: Transformers under markov data. 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 62503–62556, 2024.

[22] Shuhao Gu, Mengdi Zhao, Bowen Zhang, Liangdong Wang, Jijie Li, and Guang Liu. Retok: Replacing tokenizer to enhance representation efficiency in large language model. *arXiv:2410.04335*, 2024.

[23] Brian Lester, Jaehoon Lee, Alexander A Alemi, Jeffrey Pennington, Adam Roberts, Jascha Sohl-Dickstein, and Noah Constant. Training LLMs over neurally compressed text. *Transactions on Machine Learning Research*, 2024.

[24] Mengyu Zheng, Hanting Chen, Tianyu Guo, Chong Zhu, Binfan Zheng, Chang Xu, and Yunhe Wang. Enhancing large language models through adaptive tokenizers. 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 113545–113568, 2024.

[25] Björn Deiseroth, Manuel Brack, Patrick Schramowski, Kristian Kersting, and Samuel Weinbach. T-FREE: Subword tokenizer-free generative LLMs via sparse representations for memory-efficient embeddings. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Conference on Empirical Methods in Natural Language Processing*, pages 21829–21851, 2024.

[26] Piotr Nawrot, Jan Chorowski, Adrian Łańcucki, and Edoardo M Ponti. Efficient transformers with dynamic token pooling. *arXiv preprint arXiv:2211.09761*, 2022.

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

[28] Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We report results on multiple benchmarks in section 3 and more specific benchmarks in section 3.4.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See "limitations" in the conclusion.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: no theoretical result.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Our experiments use standard measures, experimental setup is described in section 3 and hyperparameters in appendix D. Our experiments are run on a public code, namely meta Lingua[2].

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: data are standard public datasets. We work with an open-sourced codebase for LLM, namely meta Lingua [2], and our code will be open-sourced on acceptance.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: we specify all training sets and models, including pretraining data. All details about our scaling laws and the downstream performances are specified and standard.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

We report 95% confidence interval using bootstraping in all tables.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   We use the number of flops in the x-axis of our scaling laws and specify budgets of all experiments. Section 3 we specify the type of GPUs, we are using

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   We read the ethics guidelines and did not violate anything.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: the paper is rather technical than application oriented. Though the improvements on rare languages can be positive societal impacts. We see no potential negative societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: N/A

    Justification: No models or data are released.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All assets are cited and properly credited in appendix.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: Code will be release at later date.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: Our work is about the training of LLM, so LLM are at the core of the present research. We use standard, open-sourced methodologies for evaluating LLM, using [2]. There is no other usage of LLM.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.