BYTE LANGUAGE MODELS: SCALING, EMERGENT ABSTRACTIONS, AND INFORMATION ALLOCATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Tokenizer-free language models remove the inductive bias of fixed tokenizers by modeling text directly as bytes, but the resulting longer sequences substantially increase computation and eliminate explicit text abstractions. We ask whether this additional computation can be useful, and whether standard Transformers can learn the abstractions that tokenization provides. We study these questions on Transformers without specialized tokenization-related architectures. With token-superposition training and hash embeddings, byte Transformers consistently outperform subword Transformers as model size scales. We further find that byte Transformers build local text abstractions as external tokenizers: a set of segmentation-like positions are used to collect local context representations, and restricting up to 25% of intermediate layers to these local representations preserves downstream performance. Finally, these learned structures induce highly non-uniform generation difficulty, with uncertainty concentrated near local structure boundaries; exploiting them for speculative decoding yields $3.4\times$ more accepted tokens than in subword Transformers.

1 INTRODUCTION

Tokenizer-free language models, or byte language models, aim to remove externally imposed tokenizers by modeling text directly as byte sequences (Xue et al., 2022; YU et al., 2023; Slagle, 2024; Tsiamas et al., 2025; Shah et al., 2026). This formulation provides a universal input space and a new joint learning setting. However, it also introduces two main challenges. First, byte sequences are longer than subword sequences. If a typical subword contains K (usually around 4) bytes, a byte model must process K times more sequence positions. Second, prior studies suggest that effective language modeling benefits from prior abstractions over raw text. Directly modeling bytes can therefore underperform tokenized models, despite having a more general input representation.

Recent works (BLT Pagnoni et al., 2025, H-Net Hwang et al., 2026, ByteFlow Deng et al., 2026) develop new network architectures with built-in tokenization mechanisms. For example, in BLT, local encoder layers first encode byte streams into variable-sized chunks, then global layers operate on the chunks, and local decoder layers decode chunks back to bytes. While these architectures demonstrate the promise of joint tokenization, they often need specialized training and optimization strategies, making it less clear how much of the observed gains are from the joint formulation. In this work, we retain the *flat* Transformer architecture (without the explicit local-global-local hierarchy) and investigate the two challenges underlying byte-level language modeling: *Can the additional computation caused by longer sequences be beneficial, and how does a Transformer recover useful text abstractions directly from bytes?*

We study the scaling behavior of byte Transformers, and the main findings are: 1) with two modeling ingredients (token-superposition training (Peng et al., 2026) and hash embeddings), byte Transformers consistently outperform subword Transformers when scaling the number of parameters (Figure 1a). 2) Byte models remain effective as training data continues to scale. Given a fixed FLOPs budget (Figure 1c), the optimal proportion between data and parameters is around 1.5 to 3 times that of subword models. These results motivate a different perspective on sequence length: rather than treating longer sequences as a defect, they can provide additional computation under the same parameter budget, which is a useful scaling axis for language models. Further evaluations on downstream text-only and vision–language tasks show that byte Transformers significantly improve performance on

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tasks that require fine-grained perception (around 40% relative improvements over subword Transformers on CUTE scores, 20% on OCRBench (Liu et al., 2024)).

We next investigate how a byte Transformer develops the abstractions that tokenization provides explicitly. By studying an input-copying task on byte models, we find that copying actions need to attend only to certain segmentation positions (not necessarily whitespace), where information from local text pieces can be read out. Based on this finding, we further constrain a range of intermediate layers (up to 25% of all layers) to access only these locally aggregated representations, resembling a global modeling stage over learned local structures. The resulting model retains downstream performance without additional training. This suggests that the apparent hierarchy of local abstraction and global reasoning is not merely imposed by tokenizer-free architectures such as BLT and H-Net, but can also emerge implicitly in the byte Transformer.

Finally, we study whether these learned local structures can be exploited during inference. By plotting the per-byte loss distribution, we find that byte Transformers have much higher variance than subword models: many byte positions have near-zero loss (in bits per byte), while substantially larger losses concentrate near the boundaries of the learned local structures. This observation suggests that the extra generation steps are not uniformly expensive in terms of model uncertainty. We therefore apply speculative decoding to construct local structures during inference (Wang et al., 2024; Kallini et al., 2026), and show that a drafting model on byte Transformers can produce $3.4\times$ more accepted tokens than on subword Transformers.

To summarize, our results support a simple view of tokenizer-free language modeling: longer byte sequences provide useful additional computation. At the same time, the model can learn useful local abstractions internally rather than relying on a fixed tokenizer. This perspective suggests sequence length, learned abstraction, and computation allocation as closely connected dimensions for future language-model design.

2 Scaling Byte Transformer Language Models

We first ask how the additional tokens in byte sequences affect model quality under parameter and compute constraints. We then examine which task capabilities benefit from byte resolution and how shared depth and sparse capacity change these trade-offs.

2.1 Building Byte Models

We first establish a controlled comparison between byte- and subword-level Transformers. Both models are trained from scratch with a mixed attention model. [1] The byte model uses a vocabulary of size 256, while the subword model uses a $49,152$ token vocabulary from the cosmo2-tokenizer (Allal et al., 2025). We use "token" to refer to both byte and subword, depending on the model in context.

Directly applying the standard training recipe to bytes is suboptimal. Byte sequences contain substantially more tokens, while each byte token carries less local semantic context than a subword token. We find the following two ingredients are effective for training byte models.

First, we use token-superposition training (TST; Peng et al., 2026) to improve learning efficiency from the longer byte sequences. TST averages four consecutive token embeddings and predicts the corresponding next four-token bag. Applying TST during the first 30% of training provides $4\times$ token exposure in this stage, while reverting to standard next-token prediction for the remaining 70% results in $1.9\times$ total text exposure under the same number of training tokens (see Appendix A.2 for details).

Second, we augment byte embeddings with hash embeddings as (Pagnoni et al., 2025). The model learns representations for n-grams of bytes and combines them with byte embeddings, allowing each byte token to incorporate information from nearby multi-byte patterns (Appendix A.3). Similarly,

[1] We generally follow the setting of Qwen3.5 (Qwen Team, 2026), which interleaves Gated DeltaNet (Yang et al., 2025) and Gated Softmax Attention (Qiu et al., 2025) in a $3:1$ ratio. We occasionally call the architecture *flat* Transformer, in contrast to architectures with an explicit local-global-local hierarchy.

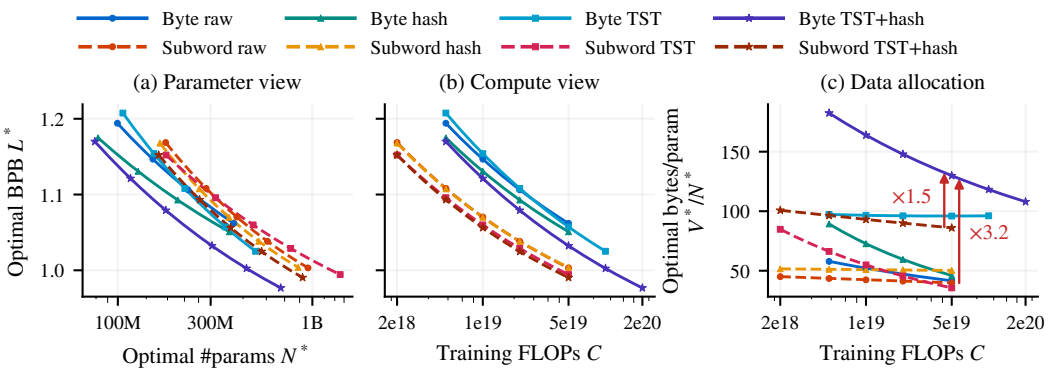

Figure 1: Pareto frontiers of compute-optimal models. (a) BYTE has lower optimal BPB $L^\star$ at the same optimal parameter count $N^\star$, while (b) SUBWORD has lower $L^\star$ at the same training compute C. (c) BYTE optima generally allocate more source bytes per parameter. Markers indicate fitted optima at the evaluated compute budgets (see details in Appendix C).

we increase the convolution width of gated DeltaNet from 4 (for subword models) to 16 (for byte models), to build a broader local receptive field.

Unless otherwise specified, byte models apply TST and hash embeddings (denoted by BYTE), while subword models don't (denoted by SUBWORD). We later ablate these components to separate the contribution of the training recipe from that of input granularity.

2.2 PARAMETER AND COMPUTE EFFICIENCY UNDER ISOFLOPS

A longer byte sequence can be viewed either as a computational burden or as an opportunity to allocate more computation to a model with the same number of parameters. To distinguish these effects, we study how byte and subword Transformers scale along three coupled axes: model size, data exposure, and training compute.

Following the standard practice of model scaling analyses, we train families of byte and subword models with different training FLOPs per token M and cumulative training data volume V (measured in bytes consumed during training), record their validation loss L (measured in bits per byte, BPB), and fit the loss L as function of M and V. We adapt the Chinchilla law (Hoffmann et al., 2022),

$$L(M, V) = E + AM^{-\alpha} + BV^{-\beta}, \tag{1}$$

where α, β, E, A, B are fiting coefficients. We also denote total training FLOPs $C = MD$, where D is the amount of training data in tokens. [2]

Following Hoffmann et al. (2022), we compare model performance in the *compute-optimal* manner. Given a FLOPs budget C, we collect loss L of models with various sizes N (or equally, M) under C, fit a curve for the obtained (N, L) pairs, and the optimal parameter count under the FLOPs budget $N^\star(C)$ is located at the minimum loss $L^\star$ (see Figure 6 and Appendix C for details). Similarly, we obtain the optimal training byte volume $V^\star(C)$. By varying C, we trace the $L^\star$–$N^\star$ and $L^\star$–C Pareto frontiers in Figure 1.

BYTE has lower optimal loss at the same optimal parameter count The $L^\star$–$N^\star$ Pareto frontiers (Figure 1a) of BYTE consistently below SUBWORD. It suggests that Transformers can transform additional computation to performance gains. Meanwhile, Figure 1b shows a different story: under the same FLOPs budgets, SUBWORD is always better. We defer discussion on scaling along computation to Section 2.4.

BYTE optima allocate more source bytes per parameter. Compared with SUBWORD, at the same training-compute budgets, the fitted optima select smaller BYTE models and generally allocate

[2] A subword token or TST bag can represent multiple source bytes, so we account for this using $V = sD$, with varying average source bytes per token s.

Table 1: BYTE consistently outperforms SUBWORD on fine-grained tasks across model scales.

	CUTE (%; ↑)		OCRBench (%; ↑)		TextVQA (%; ↑)	
Size	SUBWORD	BYTE	SUBWORD	BYTE	SUBWORD	BYTE
200M	69.89	**99.19**	27.10	**35.10**	40.62	**45.08**
400M	70.78	**99.18**	27.40	**37.40**	43.82	**48.37**
700M	73.04	**99.52**	29.90	**34.50**	45.34	**50.72**
1B	76.44	**99.56**	31.50	**39.40**	47.50	**52.36**
3B	80.83	**99.93**	34.70	**42.40**	50.76	**55.39**

more (3.2×) source bytes per parameter (Figure 1c). It implies that BYTE can reach lower BPB if training data continues to scale.

TST and hashing embeddings provide larger combined gains for BYTE. We ablate TST and hashing at matched depth and training compute within each family. Combining them consistently improves BPB over raw training, with larger gains for BYTE.

2.3 FINE-GRAINED CAPABILITIES IN TEXT-ONLY AND VISION–LANGUAGE TASKS

We next examine whether the modeling advantages of byte-level representations translate into downstream text-only and vision-language tasks. [3] [4] We evaluate word manipulation with CUTE (Edman et al., 2024), and fine-grained OCR tasks with OCRBench and TextVQA (Singh et al., 2019).

The results in Table 1 show that BYTE has a clear advantage in word manipulation capabilities over SUBWORD. We think the better fine-grained modeling ability of BYTE comes from stronger perception of inputs and stronger ability to generate byte-level sequences. We particularly expect better alignment among modalities in future byte language model-based multimodality learning.

We additionally evaluate both model families on broader standard text-only (21 tasks, Table 9) and vision-language benchmarks (5 tasks, Table 13) in Appendix D. On text tasks, BYTE model consistently outputperforms SUBWORD, due to more computation per parameter. On vision-language tasks, BYTE is on par with SUBWORD.

2.4 CONTROLLING COMPUTATION WITH SHARED DEPTH AND SPARSE EXPERTS

The preceding results reveal a compute–quality trade-off between BYTE and SUBWORD. We therefore examine two complementary ways to shift this trade-off (Figure 2): increasing SUBWORD computation through shared depth, and reducing BYTE's activated computation at a given parameter capacity with sparse mixture-of-experts (MoE). Using the main model suite, we evaluate four-pass SUBWORD models and BYTE MoE models under the training schedules of their dense counterparts; architecture details are provided in Appendix A.

Increasing computation through shared depth. We follow the setting of LoopFormer (Jeddi et al., 2026) to increase SUBWORD computation with four shared backbone passes. It closes most of the BPB gap to BYTE in both the parameter and compute views, showing that repeated depth is an effective way to scale computation for subword models. The looped SUBWORD models also achieve consistently higher QA RC than the standard dense BYTE models across the evaluated scales.

Reducing activated computation with sparse experts. Sparse MoE (DeepSeek-AI et al., 2025) consistently improves BYTE BPB in both the parameter and compute views, showing that sparse activation is an effective way to reduce computation while retaining larger model capacity. No-

[3] For Section 2.3 and 2.4, we use a separate *main model suite* spanning 200M–3B sizes, instead of the *IsoFLOPs sweep* settings in Section 2.2. At each parameter scale, BYTE and SUBWORD use the same number of updates and approximately the same source-text exposure. See Appendix B for detailed settings.

[4] We train vision-language models in the LLaVA style (Liu et al., 2023) with backbone language models replaced by BYTE. Training details are given in Appendix B.3.

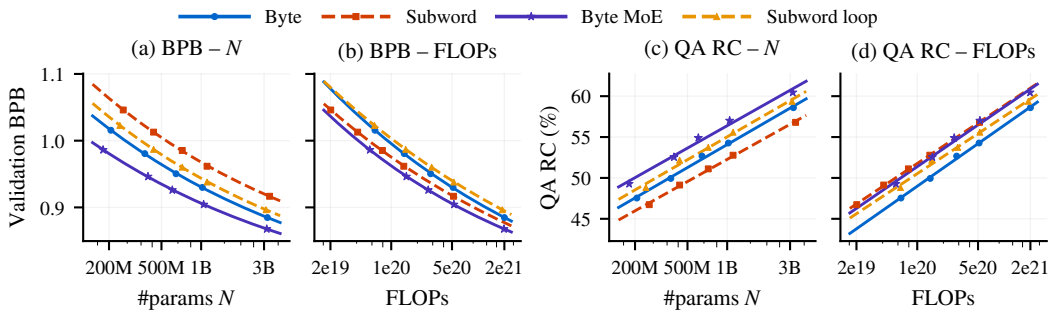

Figure 2: Compute scaling with shared depth and sparse experts. Four-pass SUBWORD closes most of the BPB gap to BYTE with higher QA RC, while sparse MoE lowers BYTE BPB with less activated computation. Markers are observations and curves are fits.

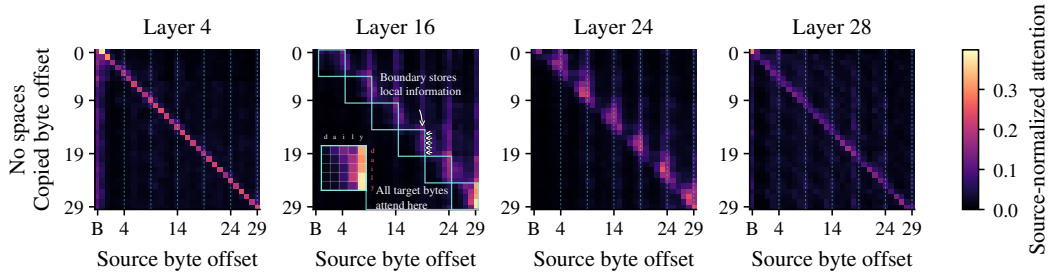

Figure 3: **BYTE-1B attention maps during copying.** Rows denote copied target bytes and columns denote source bytes from the previous occurrence; each map shows target-to-source attention. All test strings have internal spaces removed. Blue boxes indicate original word chunks, and blue dashed lines mark their final bytes. In intermediate layers, attention concentrates near chunk boundaries, and multiple subsequent target bytes repeatedly attend to the same aggregated positions, indicating the formation of meaningful local representations.

tably, the 1B BYTE MoE achieves lower BPB than the 3B dense SUBWORD model while reaching comparable QA RC.

Together, these results show that the byte–subword modeling gap depends strongly on how computation is allocated. Increasing computation through shared depth allows SUBWORD models to recover much of the BPB gap, while sparse activation improves the compute efficiency of BYTE models at comparable parameter capacity.

3 EMERGENT LOCAL ABSTRACTIONS IN BYTE TRANSFORMERS

Tokenization provides explicit local abstractions that can organize text into useful computational units. In this section, we investigate whether similar structures can emerge implicitly in byte Transformers without an explicit tokenizer or hierarchical architecture. We study this question through two complementary experiments. First, we examine how byte models organize information during prediction using an input-copying task. Second, we evaluate whether the representations revealed by this analysis can support alternative computation patterns through a frozen layer intervention.

3.1 COPYING AS A PROBE OF LOCAL INFORMATION STRUCTURE

To probe how text semantics are abstracted and accessed in models, we study the copying task by prompting the model with "Your task is to copy the string: X. Now copy it: X", where X is a short string composed of multiple meaningful English words. We evaluate BYTE-1B by analyzing attention patterns during the copying process, focusing on how different layers access the source text when generating the copied sequence (see Appendix E.1 for details).

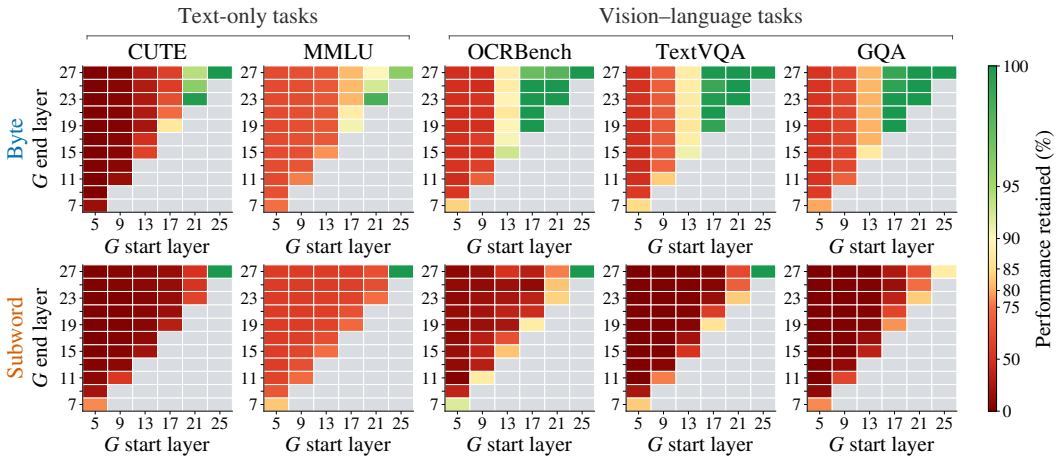

Figure 4: **Performance retention under frozen layer intervention.** Each cell shows downstream performance relative to the uncompressed baseline for a compressed layer interval $G = [a, b]$. BYTE results are shown in the top row and SUBWORD results in the bottom row. 100% indicates performance at or above the uncompressed baseline.

As shown in Figure 3, BYTE-1B exhibits a clear attention aggregation pattern during copying. In particular, attention in intermediate layers becomes concentrated on a small set of source positions that are repeatedly accessed by multiple consecutive target bytes. This aggregation occurs even without any whitespace in inputs, indicating that it does not simply arise from explicit separators. [5] The repeated reuse of these positions suggests that the byte model internally develops a mechanism for consolidating local information into representative states, providing an implicit structure over the byte sequence.

3.2 TESTING LOCAL ABSTRACTIONS WITH A FROZEN LAYER INTERVENTION

The copying task reveals that multiple byte predictions can repeatedly access a small set of locally aggregated source representations. However, attention patterns alone do not establish whether these representations are sufficient for subsequent computation. We therefore introduce a frozen layer intervention that forces a contiguous range of intermediate layers to operate on a shorter sequence of locally aggregated representations, while preserving full byte-resolution states before and after the intervention. No model parameters are updated.

Concretely, following the external tokenizer applied in BLT (Pagnoni et al., 2025), we partition the text sequence into local spans using a frozen, architecturally matched 200M BYTE model and compress non-image text tokens (for vision–language tasks) by approximately $4\times$. We treat the selected intermediate layers as a temporary global stage (like the global model in BLT): each local span is mean-pooled into a single representation, processed at the compressed resolution, and then propagated back to the original byte-resolution sequence. This intervention introduces a coarse-grained computation stage into the pretrained BYTE-1B model without modifying its parameters or retraining it. More details and results are in Appendix E.2.

We apply the same intervention to BYTE-1B and SUBWORD-1B across several downstream tasks, sweeping the start and end layers of the compressed global stage G. As shown in Figure 4, BYTE exhibits a clear recovery in performance when the global stage is placed sufficiently late in the network, with the trend becoming apparent for intervals starting around layer 17. More importantly, a stable compressible region emerges in the later layers: for $G = [21, 27]$, 7 of the model's 28 layers (25%) operate only on the compressed local representations, while downstream performance remains close to the uncompressed baseline across tasks. In contrast, applying the same intervention to the subword model causes substantial degradation across nearly all layer intervals. This contrast

[5] Whitespaces may provide shortcuts during the aggregation: model can learn to utilize repeatedly appearing tokens, ablation of separators is in Figure 9.

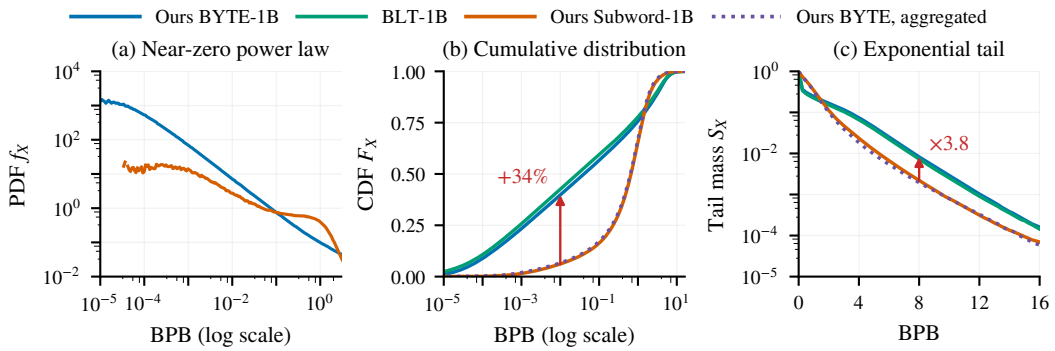

Figure 5: Token-span averaging largely closes the BPB distribution gap between BYTE and SUBWORD. The BPB distribution of BYTE has more near-zero and high-BPB mass. All panels are byte-weighted. The dotted curve averages fixed BYTE losses over SUBWORD token spans, preserving mean BPB on complete spans (Appendix F.3).

suggests that the robustness to an intermediate compressed global stage is a property of the byte model rather than a generic consequence of pooling hidden states.

4 Information Allocation Along Text Sequences

Locally aggregated representations can support part of BYTE's computation on a shorter sequence (Section 3), while the model still predicts every byte separately. These additional modeling steps raise the question of where the extra computation (Section 2.2) is useful and where it could be reduced. For a given text source, deterministic, reversible tokenizations preserve total entropy (Proposition 1), so finer granularity distributes the same source information over more tokens. We therefore examine how BPB varies across these tokens, how token boundaries explain the distribution gap, and whether small drafts can handle the low-BPB continuations.

4.1 An Empirical BPB Distribution Law

To compare local modeling difficulty across granularities, we use *byte-weighted BPB*: the random variable X gives the BPB at a uniformly sampled text byte. Following the BPB definition in Appendix D.1, a target token T_i spanning m_i original text bytes has BPB score x_i,

$$x_i = -\frac{\ln p_\theta(T_i \mid \mathbf{T}_{<i})}{m_i \ln 2}. \tag{2}$$

In the distribution statistics, each token's BPB x_i has weight m_i, accounting for differences in token length. Our main comparison uses the 1B BYTE and SUBWORD models, and we extend the analysis to public models.

The BPB density f_X exhibits a near-zero power law and an exponential tail over finite ranges, connected by an empirical transition:

$$f_X(x) \approx \begin{cases} k_{\text{pow}} x^{\alpha}, & \epsilon < x \le u_1, \\ f_X^{(\text{trans})}(x), & u_1 < x \le u_2, \\ k_{\text{exp}} e^{-x/\beta}, & u_2 < x \le v. \end{cases} \tag{3}$$

Here α is the near-zero power-law exponent and β is the exponential decay scale. The fitted ranges are $(\epsilon, u_1]$ and $(u_2, v]$; the transition $f_X^{(\text{trans})}(x)$ and values outside the fitted ranges remain empirical.[6] Let F_X and S_X denote the CDF and survival function of X, with $S_X(x) = 1 - F_X(x) = \Pr(X > x)$ (Figure 5c). We set the normalization coefficients k_{pow} and k_{exp} so that each fitted density integrates to the measured probability mass in its fitting interval:

$$k_{\text{pow}} = \frac{F_X(u_1) - F_X(\epsilon)}{\int_\epsilon^{u_1} t^{\alpha}\, dt}, \qquad k_{\text{exp}} = \frac{F_X(v) - F_X(u_2)}{\int_{u_2}^{v} e^{-t/\beta}\, dt}. \tag{4}$$

[6] Appendix F.2 gives the evaluation settings, fitting procedure, and finite-range diagnostics (Figure 12).

Byte distributions place more mass at both low and high BPB. Near zero, byte densities approach $1/x$, with fitted power-law exponent $\alpha \in [-0.97, -0.91]$, whereas subword densities are shallower, with $\alpha \in [-0.8, -0.6]$ (Figure 5a–b). In the tail, byte models have slightly larger decay scales β and more high-BPB mass overall than subword models (Figure 5c). Together, these patterns show that BYTE has more probability mass at both low and high BPB than SUBWORD (Figure 5b–c; Table 2). We also verify this law in public byte models, including BLT and H-Net, and widely used subword models. The byte models have nearly overlapping BPB distributions (Figure 11; Table 15).

Table 2: Byte models place more probability mass near zero and at high BPB in their native distributions (Section 4.1). Values are percentages of retained text bytes. Figure 11 shows the full BPB distributions.

Model	$F_X(10^{-3})$	$F_X(0.1)$	$S_X(4)$	$S_X(8)$
Ours BYTE-1B	23.3	56.6	7.43	0.822
BLT-1B	25.7	59.2	6.62	0.730
H-Net 1-stage XL	22.5	55.8	7.53	0.878
H-Net 2-stage XL	23.1	56.2	7.48	0.867
Ours SUBWORD-1B	1.5	16.3	2.35	0.218
Llama-3.1-8B	6.5	27.2	1.18	0.090
Qwen3-8B	10.5	29.3	1.91	0.204
Qwen3.5-9B	5.3	24.9	1.44	0.133

Token boundaries explain the distribution gap. We aggregate BYTE BPB by averaging within each token span defined by a subword tokenizer. The aggregated distribution (dotted lines in Figure 5b–c) closely matches the native SUBWORD distribution.[7] This result indicates that the distribution gap largely reflects how information is grouped into tokens. Subword tokenization provides a statistical prior for this grouping: it combines high-BPB bytes with predictable continuations into fewer tokens, reducing near-zero BPB mass while retaining the source's total information (Proposition 1).

Byte modeling can allocate more computation to difficult tokens. For the same total source information, BYTE uses multiple modeling steps within a span that SUBWORD handles in one step, retaining high-BPB bytes as separate tokens (Figure 5c). This supports the argument in Section 2.2 that longer sequences can scale computation without increasing model size.

4.2 DRAFTING LOW-BPB CONTINUATIONS

Maintaining a fixed byte granularity allows more computation for high-BPB tokens, but also adds modeling steps for the many low-BPB continuations (Figure 5; Table 2). We therefore examine whether speculative decoding (Leviathan et al., 2023) can reduce this overhead.

We use existing 200M and 50M models as drafts and same-family 1B models as targets for both BYTE and SUBWORD. We do not train specialized drafts. On the validation dataset, we compare accepted bytes per target forward pass in an oracle simulation on the given evaluation text. Appendix G specifies the protocol.

Higher byte acceptance rates compensate for shorter token spans. At both draft sizes, BYTE accepts about $3.4\times$ as many tokens per target forward pass as SUBWORD. Accepted bytes per forward are nearly identical with 200M drafts and slightly higher for BYTE with 50M drafts (Table 3). The 200M drafts yield longer accepted text per forward in both families.

These acceptance lengths suggest that many additional byte tokens can be verified together, reducing the need for separate sequential target calls. The comparable text coverage of the two families at both draft sizes suggests that joint verification could offset much of the extra sequential overhead introduced by byte granularity.

[7] We also ablated boundaries from the Qwen3 and Qwen3.5 tokenizers, which yielded similar aggregated distributions (Figure 13; Appendix F.3).

Table 3: Higher byte acceptance rates offset shorter token spans at both draft sizes, giving comparable accepted text per target forward pass. Targets are same-family 1B models, evaluated by an oracle simulation on the given evaluation text (Appendix G.1). Forward denotes a simulated target verification pass; counts include only accepted draft tokens (Appendix G.2).

	200M draft model		50M draft model	
	BYTE	SUBWORD	BYTE	SUBWORD
Acceptance rate (%)	90.6	74.1	84.6	62.0
Accepted tokens / forward	9.527	2.831	5.471	1.618
Byte / Subword ratio	3.366×		3.381×	
Bytes / accepted token	1.000	3.364	1.000	3.159
Accepted bytes / forward	**9.527**	**9.523**	**5.471**	**5.111**

5 RELATED WORK

Tokenizer-free language models. Tokenizer-free language models operate directly on fine-grained inputs such as bytes or characters, avoiding a separately constructed subword vocabulary. ByT5 (Xue et al., 2022) studies byte-to-byte pretraining, while MambaByte (Wang et al., 2024) models byte sequences with selective state-space models. Other works (Clark et al., 2022; YU et al., 2023; Pagnoni et al., 2025; Hwang et al., 2026), introduce hierarchical mechanisms that compress or aggregate fine-grained inputs for coarser-grained computation. In contrast, we study byte language modeling with Transformers with hybrid attention that preserves byte resolution throughout the network.

Compute scaling and allocation. Model scale and computation can be varied independently through tokenization and architectural design. Compute Optimal Tokenization (Limisiewicz et al., 2026) studies how token granularity changes the relationship between raw data, sequence length, and compute. Repeated-depth models such as LoopFormer (Jeddi et al., 2026) increase computation while sharing parameters, whereas sparse MoE models such as DeepSeek-V3 (DeepSeek-AI et al., 2025) increase parameter capacity without proportionally increasing activated computation. Building on these directions, we use shared depth to increase SUBWORD computation and sparse experts to reduce BYTE activated computation at a given parameter capacity, examining how computation shapes the byte–subword modeling trade-off.

6 DISCUSSION AND CONCLUSION

Our results provide a complementary perspective to recent byte-level approaches, which introduce explicit hierarchical or learned tokenization mechanisms to control the cost of long byte sequences. Using a flat Transformer, we show that this sequence-length overhead can instead provide useful computation: byte models benefit from scaling and favor substantially more training data per parameter under a fixed FLOP budget. Moreover, although we do not impose a hierarchy, byte Transformers develop segmentation-like positions that aggregate local text information, and intermediate layers can exploit these representations without loss in downstream performance. This connects flat byte Transformers to hierarchical architectures and suggests that local abstraction and global modeling can emerge implicitly from byte-level training. These learned structures also have inference-time consequences, as prediction difficulty concentrates near their boundaries and can be exploited by speculative decoding. Together, our results suggest that sequence length, learned abstraction, and computation should be viewed as coupled design dimensions, motivating future work on adaptive computation and learned computational granularity for byte language models.

Limitations Our study is limited to vanilla Transformer backbones and the evaluated training and task scales; whether the observed scaling behavior and emergent local structures generalize to other architectures, larger models, and broader modalities remains open.

Statement on the Use of Generative AI

In this work, we used generative AI tools to assist with experimental design, method implementation and debugging, training and mathematical exposition. We also used these tools for literature retrieval, figure and table preparation, and manuscript drafting and editing. We take full responsibility for the final manuscript and associated artifacts, including AI-assisted text, code, analyses, and claims.

Reproducibility Statement

We provide model configurations, training hyperparameters, and evaluation settings for the main model suite and IsoFLOPs sweep in Appendices A–D. Settings for BPB distribution analyses and experiments on draft acceptance and local information structure are detailed in Appendices E–G.

Ethics Statement

To the best of our knowledge, this work raises no specific ethical concerns.

References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlícek, Vaibhav Srivastav, et al. Smollm2: When smol goes big—data-centric training of a fully open small language model. In *Second Conference on Language Modeling*, 2025.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5/.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan

Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Chunyuan Deng, Sanket Lokegaonkar, Colin Lockard, Besnik Fetahu, Nasser Zalmout, and Xian Li. Byteflow: Language modeling through adaptive byte compression without a tokenizer. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=GhJIa921j7.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan SU, Markus Kliegl, ZIJIA CHEN, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan (Celine) Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-0865. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/24a8968affe71ffe4067d022b9d16566-Paper-Datasets_and_Benchmarks_Track.pdf.

Lukas Edman, Helmut Schmid, and Alexander Fraser. CUTE: Measuring LLMs' understanding of their tokens. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3017–3026, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.177. URL https://aclanthology.org/2024.emnlp-main.177/.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, Rongrong Ji, Caifeng Shan, and Ran He. Mme: A comprehensive evaluation benchmark for multimodal large language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-4899. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/d79a27cf2772fe00be7f341efc0eb517-Paper-Datasets_and_Benchmarks_Track.pdf.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: A standard for language model evaluations. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5020–5048, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.282. URL https://aclanthology.org/2025.findings-naacl.282/.

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, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=iBBcRUlOAPR.

Drew A. Hudson and Christopher D. Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust

(eds.), *International Conference on Learning Representations*, volume 2026, pp. 149273–149313, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/f1aa53cac69ef6980ca4a911ffcf278b-Paper-Conference.pdf.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 47529–47546, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/4edfc7fb472789e31de455f8735ecbc3-Paper-Conference.pdf.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Julie Kallini, Artidoro Pagnoni, Tomasz Limisiewicz, Gargi Ghosh, Luke Zettlemoyer, Christopher Potts, Xiaochuang Han, and Srini Iyer. Fast byte latent transformer. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=mo6rHb1z82.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 292–305, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.20. URL https://aclanthology.org/2023.emnlp-main.20/.

Tomasz Limisiewicz, Artidoro Pagnoni, Srini Iyer, Mike Lewis, Sachin Mehta, Alisa Liu, Margaret Li, Gargi Ghosh, and Luke Zettlemoyer. Compute optimal tokenization, 2026. URL https://arxiv.org/abs/2605.01188.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=w0H2xGHlkw.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. Mmbench: Is your multi-modal model an all-around player? In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 216–233, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-72658-3.

Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xu-Cheng Yin, Cheng-Lin Liu, Lianwen Jin, and Xiang Bai. Ocrbench: on the hidden mystery of ocr in large multimodal models. *Science China Information Sciences*, 67(12):220102, 2024.

Zongfang Liu, Shengkun Tang, Boyang Sun, Zhiqiang Shen, and Xin Yuan. Evoesap: Non-uniform expert pruning for sparse moe. *arXiv preprint arXiv:2603.06003*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 2507–2521. Curran Associates, Inc., 2022. doi: 10.52202/

068431-0182. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/11332b6b6cf4485b84afadb1352d3a9a-Paper-Conference.pdf.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. Dynamic speculation lookahead accelerates speculative decoding of large language models. In Mehdi Rezagholizadeh, Peyman Passban, Soheila Samiee, Vahid Partovi Nia, Yu Cheng, Yue Deng, Qun Liu, and Boxing Chen (eds.), *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pp. 456–467. PMLR, 14 Dec 2024. URL https://proceedings.mlr.press/v262/mamou24a.html.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Minesh Mathew, Dimosthenis Karatzas, and C.V. Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2200–2209, January 2021.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2026. URL https://arxiv.org/abs/2512.13961.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 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/.

Bowen Peng, Théo Gigant, and Jeffrey Quesnelle. Efficient pre-training with token superposition, 2026. URL https://arxiv.org/abs/2605.06546.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=1b7whO4SfY.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Arnav Shah, Junzhe Li, Parsa Idehpour, Adibvafa Fallahpour, Brandon Wang, Sukjun Hwang, BO WANG, Patrick D Hsu, Hani Goodarzi, and Albert Gu. dnaHNet: A scalable and hierarchical foundation model for genomic sequence learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=6pN2KNCspk.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Kevin 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=KEe4IUp20I.

Ioannis Tsiamas, David Dale, and Marta R. Costa-jussà. Improving language and modality transfer in translation by character-level modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20171–20187, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.988. URL https://aclanthology.org/2025.acl-long.988/.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. Mambabyte: Token-free selective state space model. *arXiv preprint arXiv:2401.13660*, 2024.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin 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/.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17084–17097. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/8df7c2e3c3c3be098ef7b382bd2c37ba-Paper.pdf.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/4904fad153f6434a7bcf04465d4be2cc-Paper-Conference.pdf.

F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, November 2003. URL https://www.rfc-editor.org/info/rfc3629/.

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 (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 78808–78823. Curran Associates, Inc., 2023. doi: 10.52202/075280-3447. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f8f78f8043f35890181a824e53a57134-Paper-Conference.pdf.

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. LMMs-eval: Reality check on the evaluation of large multimodal models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 881–916, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.51. URL https://aclanthology.org/2025.findings-naacl.51/.

A Architecture and Objectives

A.1 Tokenizers

Our byte tokenizer encodes text directly as UTF-8 without learned merges or Unicode normalization. To support document/chat control and padding within a 256-entry vocabulary, we use `0xC0`, `0xC1`, and `0xF5–0xFF` as special-token IDs. These 13 bytes values are forbidden in UTF-8 and never occur in valid UTF-8 text (Yergeau, 2003).

For subword models, we use `HuggingFaceTB/cosmo2-tokenizer` (Allal et al., 2025)[8], a pretrained byte-level BPE tokenizer with 49,152 vocabulary entries. On the pretraining dataset, it averages approximately 4.56 UTF-8 bytes per token. We use this ratio in the compute and data exposure calculations in Section 2.2 and Appendices C.

A.2 Token-Superposition Training (TST)

TST (Peng et al., 2026) lets each model input represent several tokens. We divide a token sequence $\mathbf{T}$ into non-overlapping bags of $K = 4$ consecutive tokens, forming the sequence $\mathbf{B} = (B_1, \ldots, B_{|\mathbf{T}|/K})$. Let $T_{i,j}$ be the jth token in bag B_i. Each bag is represented by the average of its token embeddings, including causal hash features when enabled.

The causal model and its single output head, with parameters θ, use the preceding bags $\mathbf{B}_{<i}$ to produce one shared token distribution $p_\theta(\cdot \mid \mathbf{B}_{<i})$ for all tokens in the next bag B_i. The loss averages token cross-entropy over all target bags (Peng et al., 2026, Eq. (3)):

$$\mathcal{L}_{\mathrm{TST}}(\mathbf{B};\theta) = -\frac{1}{|\mathbf{T}| - K} \sum_{i=2}^{|\mathbf{T}|/K} \sum_{j=1}^{K} \ln p_\theta(T_{i,j} \mid \mathbf{B}_{<i}). \tag{5}$$

The first 30% of pretraining updates use TST; the remaining 70% use ordinary next-token training on the same parameters. During TST, each four-token bag counts as one training token. With a fixed number of training tokens per update, this schedule provides $0.3 \times 4 + 0.7 \times 1 = 1.9$ times the text exposure of ordinary next-token training at the same training-token count.

A.3 Hashed Embeddings

To supply local context at the input, we augment token embeddings with hashed n-gram representations inspired by BLT (Pagnoni et al., 2025). For a sequence $x_{1:T}$ of byte or subword tokens, let $\mathcal{G}$ denote the set of n-gram groups, indexed by their lengths. The input representation is

$$e_i = \frac{1}{1 + |\mathcal{G}|} \left[E(x_i) + \sum_{n \in \mathcal{G}} E_n(\mathrm{Hash}(x_{i-n+1:i})) \right]. \tag{6}$$

Here E is the ordinary token embedding table, and E_n is a separate embedding table for length n, indexed by the shared polynomial hash Hash. When $i < n$, the missing left context is padded with zeros.

For Byte, $\mathcal{G} = \{5, 6, 7, 8\}$, with four hash tables of 32,768 slots each. For Subword, $\mathcal{G} = \{2\}$, with one hash table of 131,072 slots. Both families therefore use 131,072 hash slots in total.

Table 4 lists the settings that differ between Byte and Subword.

A.4 Dense model configuration

We scale the Qwen3.5 text architecture (Qwen Team, 2026) using the dense configurations in Table 5. Its hybrid attention combines Gated DeltaNet (GDN) and Gated Softmax Attention (Qiu et al., 2025) in a 3:1 ratio. We increase depth in four-layer increments to preserve this ratio. All models are trained from scratch with untied input and output embeddings.

[8] https://huggingface.co/HuggingFaceTB/cosmo2-tokenizer

Table 4: Architectural differences between BYTE and SUBWORD. TST and hash settings apply when the corresponding components are enabled.

Hyperparameter	BYTE	SUBWORD
Vocabulary size	256	49,152
Convolution kernel width	16	4
TST bag size / duration	4 / first 30% of updates	
Hash n-gram lengths $\mathcal{G}$	$\{5, 6, 7, 8\}$	$\{2\}$
Hash tables $\times$ slots per table	$4 \times 32{,}768$	$1 \times 131{,}072$

We use N for the number of unique active model parameters, including the output embedding. Input/hash embeddings are excluded. The difference in N between SUBWORD and BYTE models comes mainly from the larger subword output embedding, with a smaller contribution from their convolution widths.

Table 5: Dense configurations and active parameter counts N (billions). In the head-count columns, kv indicates equal key and value counts; qk indicates equal query and key counts.

Size	Layers	Width d	FFN	Head dim. GQA/GDN	GQA q/kv heads	GDN qk/v heads	BYTE N	SUBWORD N
200M	16	1,024	2,560	128/64	8/2	8/16	0.207	0.256
400M	20	1,280	3,200	128/64	10/2	10/20	0.374	0.436
700M	24	1,536	3,840	128/64	12/2	12/24	0.644	0.719
1B	28	1,792	4,480	128/64	14/2	14/28	1.022	1.109
3B	40	2,560	6,400	256/128	10/2	10/20	3.181	3.305

A.5 MOE CONFIGURATION

Hidden widths and attention configurations follow Table 5. All layers use MoE, and the expert widths in Table 6 apply to both routed and shared experts. Each layer uses 8 active experts selected from 64 routed experts, together with 1 shared expert. For MoE models, N denotes the active parameter count: it includes routers, selected routed experts, and shared experts, and excludes inactive routed experts. N_{total} counts all model parameters except hash embeddings.

Table 6: MoE configurations and parameter counts (billions).

Size	Layers	Expert width	BYTE N	BYTE N_{total}	SUBWORD N	SUBWORD N_{total}
200M	16	256	0.180	0.885	0.230	0.985
400M	20	384	0.395	2.047	0.457	2.172
700M	24	384	0.604	2.983	0.679	3.132
1B	28	512	1.044	5.361	1.131	5.535
3B	40	768	3.146	16.358	3.269	16.607

We use the auxiliary-loss-free load balancing strategy of DeepSeek-V3 (DeepSeek-AI et al., 2025). For a hidden state h, the router computes token-to-expert affinities $s_e = \sigma(w_e^\top h)$, where w_e is the router vector for expert e. Each routed expert has a bias b_e, initialized to zero. The eight experts with the largest $s_e + b_e$ form the selected set $\mathcal{K}(h)$. The selected experts are combined with weights

$$\alpha_e = \frac{2.5 s_e}{\sum_{j \in \mathcal{K}(h)} s_j}, \qquad \mathrm{MoE}(h) = f_{\text{shared}}(h) + \sum_{e \in \mathcal{K}(h)} \alpha_e f_e(h). \tag{7}$$

The bias affects expert selection, while the gating weights are computed from the original affinities s_e. After each training step, we update the biases according to

$$b_e \leftarrow b_e + \gamma \,\mathrm{sign}(\bar{n} - n_e), \tag{8}$$

where n_e is the number of token assignments to expert e across the whole training batch, $\bar{n}$ is the mean count over routed experts, and γ is the bias update speed. We use $\gamma = 10^{-3}$ and freeze the biases during the final 5% of training updates.

A.6 LOOPED MODEL CONFIGURATION

LoopFormer (Jeddi et al., 2026) repeatedly applies a shared Transformer stack, conditioning each pass on normalized time and step size. It combines next-token prediction on full and shorter trajectories with a shortcut-consistency objective to support elastic-depth inference.

Our looped SUBWORD control adopts the shared-stack recurrence with four fixed passes, retaining the baseline decoder blocks and ordinary next-token objective. Let $H^{(0)}$ denote the input token embeddings and Φ_θ the complete causal decoder stack with parameters θ. The recurrence is

$$H^{(r)} = \Phi_\theta\left(H^{(r-1)}\right), \qquad r = 1, \ldots, 4. \tag{9}$$

Both training and evaluation use this recurrence, with a single prediction head after the final pass. Training compute includes all four decoder passes and one output-head application, with forward and backward matrix operations counted as six FLOPs per participating weight per processed token. N counts each shared parameter once.

B MAIN MODEL SUITE: TRAINING DETAILS

This appendix describes training for the main model suite: the 200M–3B BYTE and SUBWORD models evaluated on CUTE, QA RC, and vision–language benchmarks in Section 2.3, together with the looped and MoE models compared in Section 2.4.

B.1 PRETRAINING HYPERPARAMETERS

All model families are pretrained on ClimbMix (Diao et al., 2025). The 16-, 20-, 24-, 28-, and 40-layer models use 50k, 60k, 70k, 80k, and 100k updates, respectively. The byte sequence length is four times the subword sequence length; with the batch sizes in Table 7, the two families process approximately the same amount of source text per update during ordinary next-token training. Because byte and subword models also use the same number of updates at each scale, their total source-text exposure under this training regime is approximately matched.

Table 7: Default pretraining hyperparameters for the main model suite.

Hyperparameter	BYTE	SUBWORD
Context length (tokens)	8,192	2,048
Global batch (tokens)	1,048,576	262,144
Base learning rate η_0	0.005	
Optimizers	Muon (matrices), AdamW (other parameters)	
Scheduler	WSD, 1% warmup, 20% 1-sqrt decay	
AdamW (β_1, β_2)	$(0.8, 0.95)$	
AdamW ϵ	10^{-10}	
AdamW weight decay	0	
Muon β_1	$0.85 \xrightarrow{\text{400 steps}} 0.97 \xrightarrow{\text{final 20\%}} 0.90$	
Muon β_2	0.95	
Muon iterations	5	
Computation precision	bfloat16	

We use μP-style hyperparameter transfer (Yang et al., 2021), with the following width and batch corrections in our Muon/AdamW optimizer (Jordan et al., 2024; Loshchilov & Hutter, 2017). Let η_0 be the common base learning rate in Table 7 and $d_0 = 768$ the reference width. Let r and c denote the numbers of rows and columns, respectively, of a parameter matrix updated by Muon. The peak learning rates for AdamW and Muon are

$$\begin{aligned} \eta_{\text{Adam}} &= \eta_0 \sqrt{2}\sqrt{d_0/d}, \\ \eta_{\text{Muon}}(r, c) &= \eta_0 \sqrt{2}\sqrt{\max(1, r/c)}. \end{aligned} \tag{10}$$

The factor $\sqrt{2}$ comes from learning-rate scaling with batch size. We neither tune nor automatically scale the global batch size, so this factor remains constant across model scales. Our pretraining hyperparameter settings are largely based on nanochat[9].

B.2 Text supervised fine-tuning

We evaluate CUTE (Appendix D.3) after SFT for both byte and subword models. Language-modeling and QA RC evaluations use base models. The SFT mixture combines Smol-SmolTalk (Allal et al., 2025), MMLU, GSM8K (Cobbe et al., 2021), and CUTE-style tasks, with loss on assistant responses. SFT starts at 0.8 of the pretraining learning rate, with 10% warmup steps and decay to zero over the final 50% of updates.

B.3 Vision–language training details

Architecture. We adapt BYTE and SUBWORD models into vision–language models (VLMs) using an identical vision–language architecture. A frozen SigLIP2 SO400M vision encoder connects to the pretrained language model through a two-layer GELU MLP projector, which maps visual representations into the language embedding space. The projected visual tokens enter the language sequence through a unified image-token interface.

Training. We use a two-stage training pipeline following the standard LLaVA recipe. Stage 1 trains only the projector on LLaVA-Pretrain, keeping the vision encoder and language model frozen, with learning rate 10^{-3} and effective batch size 256. Stage 2 jointly fine-tunes the projector and language model on LLaVA-NeXT-SFT, keeping the vision encoder frozen, with learning rate 2×10^{-5} and effective batch size 128. Both stages use AdamW, bfloat16 mixed precision, 3% warmup followed by cosine learning-rate decay, and one training epoch.

C IsoFLOPs Sweep Details

C.1 Sweep settings

The IsoFLOPs sweep builds on the pretraining setup in Appendix B.1 and uses the BPB evaluation protocol in Appendix D.1. We evaluate dense byte and subword models under four recipes: raw, hash-only, TST-only, and TST+hash. To cover the optimal model sizes at low FLOP budgets, we extend the dense configurations in Table 5 down to 8 layers. For each model size, the number of training updates is determined by the FLOP budget, replacing the prescribed update counts of the main model suite. Warmup lasts 1,000 updates instead of 1% of training; the remaining training hyperparameters, including decay over the final 20% of updates, follow Table 7. Architecture and enabled TST/hash components use the settings in Table 4. Figure 6 presents the complete IsoFLOPs sweep results across all evaluated model sizes and FLOP budgets.

C.2 Parameter, FLOP, and data accounting

Training FLOPs per token. To compare compute costs across model sizes and token granularities, we estimate training FLOPs per token as

$$M = 6N + \sum_{\ell \in \mathcal{A}} 12 H_\ell^Q d_\ell^h T + \sum_{\ell \in \mathcal{D}} 24 H_\ell^V d_\ell^k d_\ell^v. \tag{11}$$

The sums add full-attention and Gated DeltaNet recurrent operations, with layer sets $\mathcal{A}$ and $\mathcal{D}$, respectively. T is the context length in training tokens, H_ℓ^Q, H_ℓ^V are query/value head counts, and $d_\ell^h, d_\ell^k, d_\ell^v$ their dimensions. By convention, embedding lookups, hashing, bag averaging, loss evaluation, and optimizer updates are excluded from this estimate.

To convert fitted loss curves and compute-optimal allocations from M to N smoothly between trained sizes, we express both counts in terms of model width d. Substituting the architecture hyper-

[9] https://github.com/karpathy/nanochat

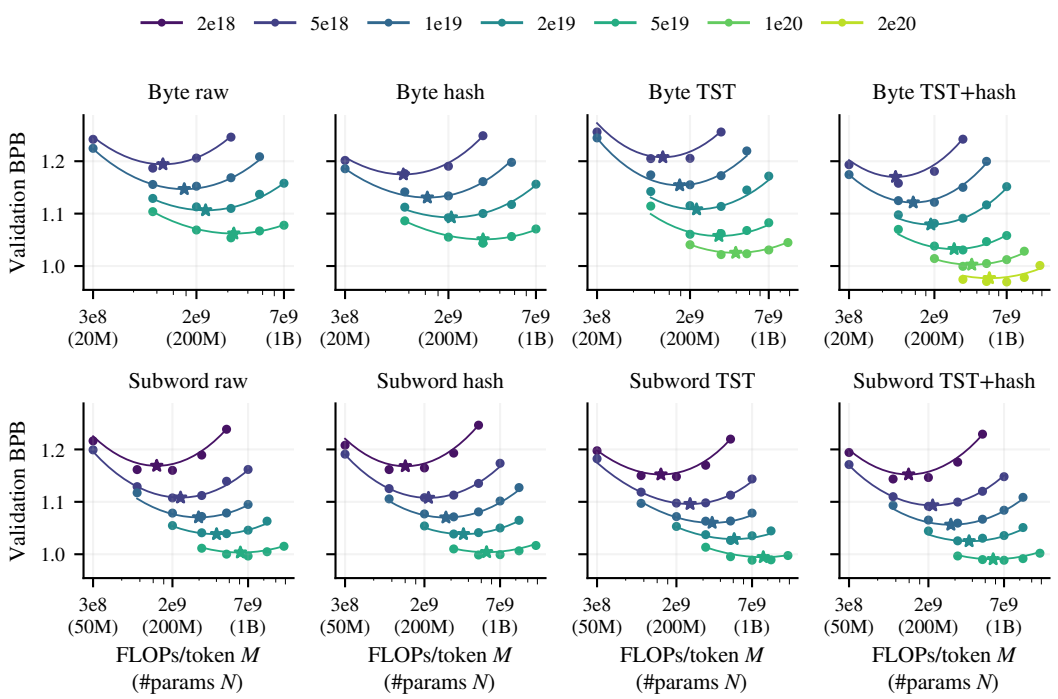

Figure 6: TST+hash lowers fitted optimal BPB in both families at shared budgets, with larger gains for bytes. Dots are measurements; curves are fixed-budget slices of $L(M, V)$, and stars are fitted optima. Colors denote training FLOP budgets.

parameters and context lengths of the discrete sweep configurations gives

$$\begin{aligned} N_{\text{byte}}(d) &= \frac{1443}{8192}d^3 + \frac{19715}{8192}d^2 + \frac{1031}{4}d, \\ N_{\text{subword}}(d) &= \frac{1443}{8192}d^3 + \frac{17411}{8192}d^2 + \frac{196615}{4}d, \\ M_{\text{byte}}(d) &= \frac{4329}{4096}d^3 + \frac{1665}{4}d^2 + 1536d, \\ M_{\text{subword}}(d) &= \frac{4329}{4096}d^3 + \frac{2025}{16}d^2 + 294912d. \end{aligned} \tag{12}$$

We then allow d to vary continuously in these expressions to obtain the mapping $N(M)$.

Source-byte exposure volume. Let D be the number of training tokens, counting each superposed bag as one token during TST. Cumulative source-byte exposure is $V = sD$, where s is the mean number of source bytes per training token, averaged over the training schedule. Ordinary next-token training uses $s = 1$ for bytes and $s = 4.561464$ for subwords. TST with four-token bags for the first 30% of updates multiplies either value by 1.9 (Appendix A.2). It increases source-byte exposure at a fixed training-token count while leaving per-token compute M unchanged.

Total training FLOPs. Combining per-token compute and source-byte exposure gives

$$C_{\text{iso}} = MD = \frac{MV}{s}. \tag{13}$$

At a fixed FLOP budget, this relation determines the source-byte exposure available to each model size.

C.3 Global fit and compute-optimal allocation

We fit Equation 1 independently to all observations of each recipe, adapting the Chinchilla law (Hoffmann et al., 2022, Section 3.3 and Appendix D.2) to per-token compute M and source bytes

V. We estimate the parameters by minimizing a Huber loss with threshold 0.001 between predicted and observed log losses. Bounded L-BFGS-B uses 500 random initializations.

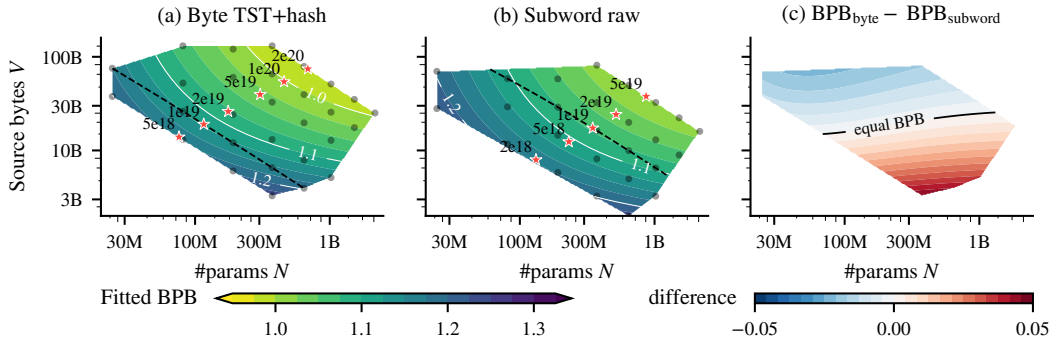

Figure 7: At matched N, V, BYTE TST+hash improves over raw SUBWORD as source-byte exposure grows. Here, N excludes the output embedding to align parameter counts between the two model families. Panels (a–b) show fitted BPB, and (c) shows their difference. Stars are fitted compute-optimal allocations; black dots are training configurations, and dashed contours mark 10^{19} FLOPs. Surfaces cover only the sampled parameter–data regions.

The interior optimum at budget C is

$$M^\star(C) = \left[\frac{\alpha A}{\beta B}(sC)^\beta\right]^{1/(\alpha+\beta)}, \qquad V^\star(C) = \frac{sC}{M^\star(C)}. \tag{14}$$

Let $N^\star(C)$ be the parameter count corresponding to $M^\star(C)$ and $L^\star(C)$ the resulting fitted loss. The interior balance $\alpha A(M^\star)^{-\alpha} = \beta B(V^\star)^{-\beta}$ yields

$$\frac{V^\star}{N^\star} = \left(\frac{\beta B}{\alpha A}\right)^{1/\beta} \frac{[M(N^\star)]^{\alpha/\beta}}{N^\star}, \qquad L^\star = E + \left(1 + \frac{\alpha}{\beta}\right) A[M(N^\star)]^{-\alpha}. \tag{15}$$

Thus the optimal data-to-parameter ratio depends jointly on the learned model/data exponents and architectural compute costs. For the reported allocations, we clip $M^\star$ to each recipe's observed size range and convert it to $N^\star$ using Equation 12. Varying C yields the $(N^\star(C), L^\star(C))$ and $(C, L^\star(C))$ Pareto frontiers in Figure 1.

Table 8: Fitted parameters of $L(M, V) = E + AM^{-\alpha} + BV^{-\beta}$ (Section 2.2).

Recipe	A	α	B	β	E
Byte raw	1054.9	0.42293	13257	0.50811	0.87358
Byte hash	300.19	0.36138	53129	0.57329	0.86467
Byte TST	2547.6	0.46305	3691.2	0.42706	0.83219
Byte TST+hash	550.58	0.39097	9343.3	0.46963	0.81537
Subword raw	2410.6	0.46131	10460	0.48918	0.85900
Subword hash	3810	0.48746	10338	0.48703	0.86609
Subword TST	229.07	0.33740	81150	0.57364	0.83360
Subword TST+hash	1825.6	0.44984	14020	0.48763	0.84645

D EVALUATION PROTOCOLS AND ADDITIONAL RESULTS

D.1 VALIDATION BITS PER BYTE

We evaluate language modeling on the small held-out perplexity validation set used in OLMo 3 training (Olmo et al., 2026)[10]. It contains 18,523 documents from C4, Dolma (books, Common Crawl, peS2o, Reddit, The Stack, and Wikipedia), ICE, M2D2 S2ORC, the Pile, and WikiText-103.

[10] https://github.com/allenai/OLMo-core/blob/main/src/olmo_core/data/mixes/v3-small-ppl-validation.txt

To fairly compare across tokenizations, we normalize next-token cross-entropy by the number of text bytes. Let $\mathbf{T} = (T_1, \ldots, T_{|\mathbf{T}|})$ be the token sequence and $\mathbf{B} = (B_1, \ldots, B_{|\mathbf{B}|})$ the UTF-8 byte of the same sequence. For a model with distribution p_θ, BPB is calculated as

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{T}; \theta) = -\frac{1}{|\mathbf{T}|} \sum_{i=1}^{|\mathbf{T}|} \ln p_\theta(T_i \mid \mathbf{T}_{<i}),$$
$$\mathrm{bpb} = \frac{|\mathbf{T}|}{|\mathbf{B}|} \frac{\mathcal{L}_{\mathrm{CE}}(\mathbf{T}; \theta)}{\ln 2}. \tag{16}$$

D.2 Question answering (QA RC)

We evaluate base-model question answering (QA) using the QA RC task group of the OLMo 3 Base-Easy suite (Olmo et al., 2026), implemented in OLMES (Gu et al., 2025)[11]. RC denotes ranked classification. Each candidate is scored separately by summing the conditional token log-probabilities of its associated continuation, using the suite's task-specific completion prompts and in-context examples.

We report QA RC as the macro-average of 21 task-family scores, expressed as a percentage. For ARC, MMLU, and Basic Skills, constituent task accuracies are first averaged with equal weight within each family; the final average then gives each task equal weight. Table 9 breaks down QA RC by task; Figure 8 extends the QA RC comparison to all model families.

Table 9: QA RC scores (%; ↑) for our models.

	200M		400M		700M		1B		3B	
Task	Subword	Byte	Subword	Byte	Subword	Byte	Subword	Byte	Subword	Byte
ARC	54.27	53.67	55.56	58.58	59.84	60.49	61.40	62.83	65.81	68.36
MMLU	29.68	29.57	31.09	31.50	32.83	33.63	34.02	35.64	37.08	38.63
CSQA	53.24	52.50	57.08	54.71	57.08	60.11	62.24	61.34	64.62	69.12
HellaSwag	47.24	51.32	52.48	56.91	57.14	62.17	60.94	65.56	67.84	72.89
WinoGrande	51.78	52.72	53.83	55.17	55.88	59.19	58.01	58.96	60.14	62.98
SocialIQA	43.24	46.11	47.08	47.54	48.87	48.26	48.87	51.02	52.76	54.76
PiQA	70.57	70.02	72.74	73.61	74.37	75.19	75.08	76.99	78.29	78.84
CoQA	67.48	68.53	71.29	72.46	73.42	75.49	74.79	76.82	81.72	82.79
DROP	27.30	28.77	28.66	29.53	31.09	31.81	30.50	32.59	34.26	37.86
Jeopardy	37.30	36.78	44.12	40.69	46.16	48.02	53.12	52.83	61.60	64.46
NaturalQs	24.81	24.72	25.69	27.00	27.39	29.18	29.82	31.61	33.31	35.68
SQuAD	66.80	71.26	71.90	76.77	76.87	80.64	78.94	84.08	85.76	90.49
SciQ	78.10	78.90	82.40	84.10	84.90	86.30	86.80	89.60	90.10	91.40
QASPER	67.08	63.32	67.08	61.76	66.14	65.52	67.08	62.38	65.52	63.32
Basic Skills	49.30	51.66	51.89	53.47	54.97	57.79	58.15	59.40	64.51	65.08
DBQA	27.12	28.65	25.38	30.19	29.42	29.04	28.85	27.88	27.88	25.58
ProtocolQA	27.78	26.85	27.78	25.93	24.07	24.07	25.93	25.93	26.85	26.85
LAMBADA	36.97	39.06	40.21	42.07	43.94	45.45	45.47	48.26	51.31	53.70
MedMCQA	27.11	28.50	27.99	29.17	28.38	30.00	29.14	30.60	30.96	32.42
MedQA	21.45	22.70	23.33	23.10	24.12	24.74	25.14	25.14	27.42	30.32
SciRIFF	72.63	72.69	74.21	74.40	76.49	79.01	74.15	80.21	85.15	84.77
Mean	46.73	47.54	49.13	49.94	51.11	52.67	52.78	54.27	56.80	58.59

D.3 CUTE: character and word manipulation

CUTE (Edman et al., 2024) tests spelling, inverse spelling, containment, and character/word insertion, deletion, swapping and substitution, including random-string variants. We evaluate 19 variants using fixed demonstration examples and greedy generation. Containment scores are adjusted from a to $\max(0, 2a - 1)$ to remove the binary chance baseline. CUTE is evaluated with models after text SFT (Appendix B.2).

Aggregate and task-level scores for main models are given in Tables 1 and 10, respectively; Figure 8 extends the CUTE comparison to all model families.

[11] https://github.com/allenai/olmes

Table 10: CUTE scores (%; ↑) for our main models.

	200M		400M		700M		1B		3B	
Task	SUBWORD	BYTE	SUBWORD	BYTE	SUBWORD	BYTE	SUBWORD	BYTE	SUBWORD	BYTE
Spelling	51.70	99.80	56.10	99.80	60.00	99.90	68.60	99.80	81.80	100.00
Spelling (random)	99.10	99.90	99.00	99.90	99.40	99.00	99.60	100.00	99.80	100.00
Inverse spelling	59.90	100.00	63.20	100.00	73.90	100.00	78.30	100.00	82.00	100.00
Inverse spelling (random)	87.30	100.00	93.00	99.90	91.20	100.00	95.60	100.00	97.60	100.00
Character containment	67.20	99.20	70.60	99.60	72.60	100.00	74.00	99.80	78.60	100.00
Containment (random)	99.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Word containment	99.80	99.00	100.00	98.80	100.00	99.60	99.80	99.80	100.00	100.00
Character insertion	17.30	99.70	19.80	99.30	21.20	99.30	23.20	99.20	31.50	99.80
Insertion (random)	79.40	99.90	80.90	99.90	84.60	100.00	92.20	100.00	95.10	100.00
Word insertion	97.60	98.10	97.90	97.50	97.30	98.70	98.30	98.60	98.70	99.80
Character deletion	24.10	99.40	24.70	99.60	30.50	99.20	32.20	99.50	41.10	100.00
Deletion (random)	95.40	100.00	95.20	100.00	96.40	99.90	98.20	100.00	99.00	100.00
Word deletion	96.70	97.20	97.80	94.30	97.80	98.70	98.40	98.70	98.10	99.70
Character swapping	4.00	99.50	3.20	99.50	6.00	99.70	9.60	99.60	18.00	100.00
Swapping (random)	59.30	99.90	51.50	99.90	57.80	100.00	73.10	100.00	89.50	100.00
Word swapping	94.20	95.50	95.90	98.00	96.50	97.90	97.50	98.00	98.80	99.50
Character substitution	17.20	99.50	19.70	99.80	22.50	99.60	25.30	99.50	32.00	100.00
Substitution (random)	80.60	100.00	78.40	100.00	81.90	100.00	89.70	99.90	94.80	100.00
Word substitution	98.10	98.10	97.90	98.60	98.10	99.40	98.70	99.30	99.40	99.80
Mean	69.89	99.19	70.78	99.18	73.04	99.52	76.44	99.56	80.83	99.93

Prompts and task examples. Each evaluation prompt presents the task instruction, four numbered question–answer demonstrations, and the test question in a single user message. The demonstrations are fixed for each task and shared by BYTE and SUBWORD; random-string variants reuse the corresponding task's instruction and demonstrations. Table 11 shows a complete spelling prompt and selected evaluation questions with their reference answers.

Table 11: CUTE evaluation prompt and task examples. The spelling prompt is shown in full; the additional examples show only the final question. Reference answers are displayed separately from the model input.

Complete spelling prompt (user message)

```
Spell out the word, putting spaces between each letter, based on the
following examples:
1. Question: Spell out the word "alphabet". Answer: "a l p h a b e t"
2. Question: Spell out the word "hello". Answer: "h e l l o"
3. Question: Spell out the word "zebra". Answer: "z e b r a"
4. Question: Spell out the word "tongue". Answer: "t o n g u e"
Question: Spell out the word "there". Respond only with the final
answer. Answer:
```

Reference answer: `t h e r e`

Task	Final question	Reference answer
Spelling (random)	`Question: Spell out the word "bdj".`	`b d j`
Inverse spelling	`Question: Write the word "t h e".`	`the`
Character insertion	`Question: Add an "l" after every "t" in "little".`	`litltlle`
Word swapping	`Question: Swap "is" and "fun" in "It is fun.".`	`It fun is.`

D.4 VISION–LANGUAGE EVALUATION

We evaluate with the `lmms-eval` framework (Zhang et al., 2025) on vision–language benchmarks covering several capabilities. DocVQA (Mathew et al., 2021), OCRBench, TextVQA, and ChartQA (Masry et al., 2022) assess OCR and document understanding; ScienceQA-IMG (Lu et al., 2022) and MMBench (Liu et al., 2025) assess knowledge-intensive visual reasoning. MME (Fu

et al., 2025), GQA (Hudson & Manning, 2019), and POPE (Li et al., 2023) assess general visual understanding, covering object hallucination, visual question answering, and perception and cognition.

OCR task examples. The OCRBench examples in Table 12 illustrate direct transcription of irregular, handwritten, and non-semantic text. The TextVQA examples require identifying the text relevant to a question within a scene.

Table 12: OCRBench and TextVQA examples with questions and reference answers. Images are omitted.

Benchmark / task	Question	Reference answer
OCRBench Irregular text	What is written in the image?	`PARLIAMENT`
OCRBench Handwriting	What is written in the image?	`strictures`
OCRBench Non-semantic text	What is written in the image?	`ntishgcwi`
TextVQA Scene text QA	What does the small white text spell?	`copenhagen`
TextVQA Scene text QA	What is the license plate number of this vehicle?	`aj52uyv`
TextVQA Scene text QA	What is the phone number listed to rent this billboard?	`648-3004`

Table 13 and Table 14 show that performance on the remaining vision–language tasks is more task-dependent than on OCRBench and TextVQA. Across both dense and MoE models, the differences between BYTE and SUBWORD are mixed at smaller scales, but generally become more favorable to BYTE as model scale or capacity increases. For the largest dense models, BYTE surpasses SUBWORD on ScienceQA-IMG, MMBench, and POPE and nearly matches it on GQA, while the largest BYTE MoE model achieves higher scores on several broader vision–language benchmarks, including ScienceQA-IMG, MMBench, MME, and GQA. Together with the consistent gains on OCRBench and TextVQA reported in Table 1, these results suggest that the clearest advantage of byte-level modeling appears on fine-grained vision–language tasks, while broader vision–language capabilities become increasingly competitive with scale and capacity.

Table 13: Vision–language evaluation for dense models.

		OCR & document understanding				Knowledge-intensive visual reasoning		General visual understanding		
Size	Model	DocVQA	OCRBench	TextVQA	ChartQA	ScienceQA-IMG	MMBench	MME	GQA	POPE
200M	SUBWORD	25.98	27.10	40.62	24.04	**42.49**	**8.16**	1164.20	**48.26**	**85.37**
200M	BYTE	**28.56**	**35.10**	**45.08**	**26.56**	37.88	2.06	**1187.69**	46.26	84.65
400M	SUBWORD	29.77	27.40	43.82	28.00	46.01	**23.11**	**1245.94**	**49.67**	82.28
400M	BYTE	**32.64**	**37.40**	**48.37**	**28.72**	**48.19**	13.48	1206.10	49.13	**82.95**
700M	SUBWORD	31.35	29.90	45.34	30.08	**50.17**	29.98	**1332.91**	**51.87**	**85.39**
700M	BYTE	**33.43**	**34.50**	**50.72**	**30.56**	47.65	**30.15**	1326.96	50.50	82.46
1B	SUBWORD	**34.38**	31.50	47.50	30.96	**54.93**	**36.68**	1327.14	**53.21**	**84.99**
1B	BYTE	34.19	**39.40**	**52.36**	**32.12**	47.89	31.01	**1344.80**	52.28	84.51
3B	SUBWORD	37.40	34.70	50.76	36.92	56.97	44.15	**1470.32**	**56.38**	85.72
3B	BYTE	**38.83**	**42.40**	**55.39**	**38.36**	**60.54**	**45.62**	1428.47	55.99	**86.13**

Table 14: Vision–language evaluation for MoE models.

		OCR & document understanding				Knowledge-intensive visual reasoning		General visual understanding		
Size	Model	DocVQA	OCRBench	TextVQA	ChartQA	ScienceQA-IMG	MMBench	MME	GQA	POPE
200M	SUBWORD	**25.87**	26.00	39.79	23.04	**50.37**	**2.66**	**1219.18**	**46.10**	**82.26**
200M	BYTE	25.85	**32.90**	**43.22**	**23.28**	39.96	1.37	1154.95	43.81	81.36
400M	SUBWORD	28.68	27.10	44.01	23.48	44.97	6.35	1265.12	47.06	**84.78**
400M	BYTE	**30.84**	**35.90**	**49.61**	**27.32**	**47.30**	**18.21**	**1292.77**	**47.61**	84.75
700M	SUBWORD	34.26	33.40	49.28	34.48	**52.85**	**35.57**	1335.96	53.65	85.17
700M	BYTE	**35.15**	**39.60**	**53.50**	**35.36**	51.36	30.41	**1341.47**	**53.78**	**86.04**
1B	SUBWORD	**36.39**	**33.40**	**50.13**	**35.88**	**56.17**	30.32	1247.24	**54.91**	86.90
1B	BYTE	33.36	32.30	42.45	34.72	54.04	**38.57**	**1390.83**	49.39	**86.96**
3B	SUBWORD	**42.17**	39.70	53.57	40.84	58.50	46.13	1469.19	57.87	**87.36**
3B	BYTE	41.87	**46.10**	**59.70**	**43.08**	**61.47**	**55.41**	**1576.49**	**59.06**	87.33

D.5 Parameter and compute efficiency across model families

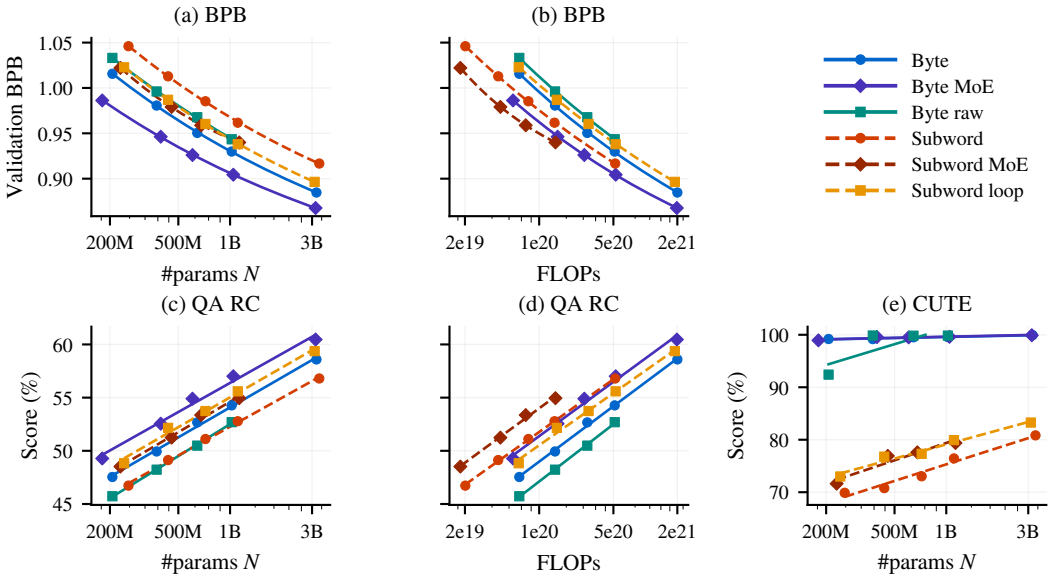

Figure 8: In the main model suite, dense BYTE achieves lower BPB and higher QA RC than depth-matched dense SUBWORD, with greater training compute. BPB and QA RC use base models, and CUTE uses SFT models. Points are observations; curves summarize trends within each family's observed range (Appendix D).

E Details of Local Information Structure Experiments

This appendix provides additional details for the local information structure experiments in Section 3. We first describe the input-copying task, including the prompt construction, input variants, and attention measurement procedure. Additional analyses with alternative separator settings are also provided. We then describe the frozen layer intervention, including local span construction, compressed global stage design, and downstream evaluation settings.

E.1 Input-copying Task Setting

Prompt and input variants. Copying requires the model to retrieve information from an earlier occurrence of the same text, making it a useful probe for studying how information is organized and accessed internally. Following the use of repeated sequences to study induction heads (Olsson et al., 2022), we inspect BYTE-1B with the prompt

```
Your task is copy the string:  X now copy it:  X
```

Both occurrences of X are supplied in a single teacher-forced causal forward pass. The string contains ten six-word English phrases; each word has exactly 5 ASCII bytes, allowing word boundaries to align across examples. For example, one phrase is `"quiet birds cross ocean waves daily"`. The original and space-free variants contain 35 and 30 bytes, respectively.

Attention measurement. For copied byte offset r, the query corresponds to the position immediately before that byte, whose hidden state predicts the next byte. The key set $\mathcal{K}$ contains the source phrase and its preceding separator position, denoted as B. Let $A^{(\ell)}_{n,h}(r,k)$ denote the raw attention weight from this query to source key k in example n, head h, and layer ℓ. With $N = 10$ examples and $H = 14$ heads, we compute

$$\bar{A}^{(\ell)}_{r,k} = \frac{1}{NH} \sum_{n=1}^{N} \sum_{h=1}^{H} A^{(\ell)}_{n,h}(r,k), \tag{17}$$

and normalize each row within the source region:

$$\widetilde{A}^{(\ell)}_{r,k} = \frac{\bar{A}^{(\ell)}_{r,k}}{\sum_{j \in \mathcal{K}} \bar{A}^{(\ell)}_{r,j}}. \tag{18}$$

Thus, each row represents the attention distribution from the query position for a copied byte token to the source sequence. Word-end guides indicate the final byte of each original word; in the space-free condition, these guides are only annotations and do not appear in the input.

Alternative separator analysis. To investigate how word separators affect attention, we compare spaces with `|`, `;`, and `and`. Only separators inside X change; the underlying phrases and surrounding instruction remain fixed. Figure 9 shows the resulting attention maps.

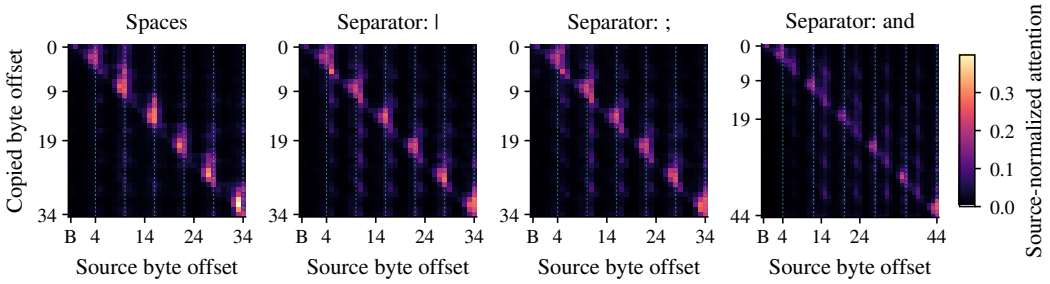

Figure 9: **Ablations on separators: Local retrieval persists with alternative separators.** Layer-24 BYTE-1B maps use the copying probe in Figure 3, with words separated by spaces, `|`, `;`, or `and`. Dashed lines mark original word ends; colors share the same source-normalized attention scale.

E.2 FROZEN LAYER INTERVENTION

Local span construction. The frozen layer intervention introduces a compressed global stage over local representations while preserving the original byte-resolution computation before and after the selected layers. Inspired by the local–global–local organization of BLT and H-Net (Pagnoni et al., 2025; Hwang et al., 2026), we identify local spans in the prompts of the 28-layer BYTE-1B and SUBWORD-1B models using a matched 200M model from the corresponding family. For a prompt token T_i, the smaller model with parameters θ computes the token loss

$$\mathcal{L}_{\mathrm{CE}}(T_i \mid \mathbf{T}_{<i}; \theta) = -\ln p_\theta(T_i \mid \mathbf{T}_{<i}). \tag{19}$$

Tokens with high loss mark patch boundaries, grouping predictable continuations into local spans.

For text-only tasks with T prompt tokens, we target $\lceil T/4 \rceil$ patches, corresponding to approximately $4\times$ compression of the text sequence. The first token always starts a patch, and we select the highest loss threshold that yields the target number of patch starts, retaining all ties.

For vision–language tasks with T non-image prompt tokens and V image tokens, we apply the same $4\times$ target ratio only to the text portion, giving a target of $V + \lceil T/4 \rceil$ patches. Each image token is kept as a singleton patch, while text patch boundaries are selected using the same loss threshold procedure. In both settings, prompt patches remain fixed during generation, and each newly generated token is appended as a singleton patch.

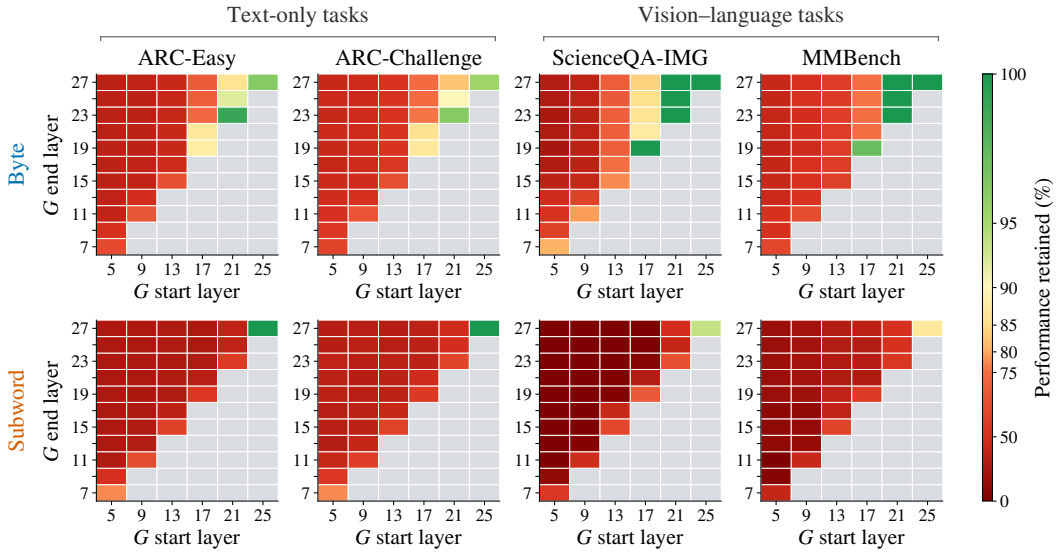

Figure 10: **Additional performance retention results under the frozen layer intervention.** Each cell reports downstream performance relative to the uncompressed baseline for a compressed layer interval $G = [a, b]$. The axes denote the start layer a, end layer b, and interval length $|G| = b - a + 1$. BYTE results are shown in the top row and SUBWORD results in the bottom row. 100% indicates performance at or above the uncompressed baseline.

Compressed global stage. Let $G = [a, b]$ denote the selected layer interval. Before layer a, the model operates at the original resolution and produces hidden states h_i. For a local span P_j, we construct a pooled representation

$$z_j = \frac{1}{|P_j|} \sum_{i \in P_j} h_i. \tag{20}$$

The original layers in G then operate on the shorter sequence of local representations. After processing, the global updates are mapped back to the original resolution through a residual reconstruction:

$$\hat{h}_i = h_i + \Delta_{j(i)}, \qquad j(i) = \max\{j : e_j \leq i\}. \tag{21}$$

Here, e_j is the final position of span P_j, and Δ_j is its representation update after the global stage. Tokens before the first completed span keep their original states.

This preserves the original byte-level states while requiring the selected layers to process only local aggregated representations.

Evaluation settings and additional results. We sweep the start layer $a \in \{5, 9, 13, 17, 21, 25\}$ and interval lengths $|G| \in \{3, 5, \ldots, 23\}$, retaining intervals with $b \leq 27$. The final layer operates at full resolution after reconstruction. No parameter updates or additional training are performed.

For text-only evaluation, we use CUTE, ARC-Easy, ARC-Challenge, and MMLU. For vision–language evaluation, we use OCRBench, TextVQA, GQA, ScienceQA-IMG, and MMBench. Results on CUTE, MMLU, OCRBench, TextVQA, and GQA are reported in Figure 4. The remaining results on ARC-Easy, ARC-Challenge, ScienceQA-IMG, and MMBench are shown in Figure 10.

F BPB Distribution Analyses

F.1 Source entropy and token granularity

We first establish a common total entropy for different tokenizations of the same text source. Here H denotes Shannon entropy in bits.

Proposition 1 (Entropy invariance across tokenizers)**.** *Let $\mathbf{B}$ be a random complete finite byte sequence with finite entropy. For two fixed deterministic tokenizers τ_1 and τ_2, define $\mathbf{T}^{(j)} = \tau_j(\mathbf{B})$ for $j \in \{1, 2\}$. If each encoding uniquely determines the original byte sequence, then*

$$H(\mathbf{T}^{(1)}) = H(\mathbf{T}^{(2)}) = H(\mathbf{B}). \tag{22}$$

Proof. Let $\mathcal{S}$ be the support of $\mathbf{B}$. For either tokenizer τ_j, unique recovery implies that $\tau_j(\mathbf{b}) = \tau_j(\mathbf{b}')$ only if $\mathbf{b} = \mathbf{b}'$. Thus τ_j is a bijection between $\mathcal{S}$ and the support $\tau_j(\mathcal{S})$ of $\mathbf{T}^{(j)}$. Let p_B be the source probability mass function and p_j the induced probability mass function of $\mathbf{T}^{(j)}$. We have $p_j(\tau_j(\mathbf{b})) = p_B(\mathbf{b})$ for every $\mathbf{b} \in \mathcal{S}$, so reindexing the entropy sum gives

$$\begin{aligned} H(\mathbf{T}^{(j)}) &= -\sum_{\mathbf{b}\in\mathcal{S}} p_j(\tau_j(\mathbf{b})) \log_2 p_j(\tau_j(\mathbf{b})) \\ &= -\sum_{\mathbf{b}\in\mathcal{S}} p_B(\mathbf{b}) \log_2 p_B(\mathbf{b}) = H(\mathbf{B}). \end{aligned}$$

□

For each complete text $\mathbf{b}$ in the source support, the same probability correspondence preserves its total information $-\log_2 p_B(\mathbf{b})$ across tokenizations.

Byte prediction exposes the intermediate prefixes inside a subword span, whereas a subword model predicts the span jointly. This source invariance motivates the comparison, but model BPB also reflects approximation error and available context.

F.2 Experimental settings for BPB distribution analysis

We compare models on a 2 MiB subset of the evaluation dataset in Appendix D.1. Subword models are evaluated with sequences of up to 4,096 tokens. Figure 11 shows the full BPB distributions.

H-Net target filtering. H-Net's highest BPB values concentrate on TAB (`0x09`), LF (`0x0A`), CR (`0x0D`), `{` (`0x7B`), and `}` (`0x7D`). H-Net were trained on FineWeb-Edu, a filtered subset of FineWeb. FineWeb's C4 filter rejects an entire document when a surviving line contains `{`, also removing any paired `}` in that document. It drops empty lines and joins retained lines with a single LF, while text extraction collapses whitespace such as TAB and CR. These preprocessing choices may explain the extreme BPB values through reduced training exposure to braces, TAB, CR, and repeated line breaks.

To limit the influence of these formatting differences on the distribution comparison, we exclude the five target characters from H-Net's distribution statistics while retaining them in the context. H-Net thus retains 98.5% of the full evaluation set.

Finite-range density fits. To identify the BPB ranges where each density law holds, we use the local power-law exponent and exponential decay scale shown in Figure 12:

$$\alpha_{\text{local}}(x) = \frac{d \ln f_X(x)}{d \ln x}, \qquad \beta_{\text{local}}(x) = -\left[\frac{d \ln f_X(x)}{dx}\right]^{-1}. \tag{23}$$

For the power law, we select the widest stable interval spanning at least 1.5 decades; for the exponential tail, we select the longest stable core spanning at least 4 BPB. Both require global and local log-density RMS errors at most 0.15, and local power-law exponents must also lie within 0.15 of the fitted α.

Table 15 reports the resulting fitting ranges, α, and β.

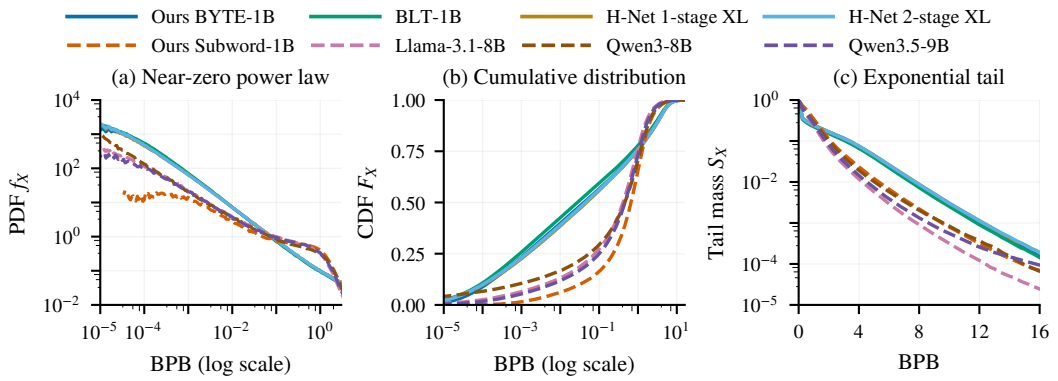

Figure 11: Byte models show more near-zero and high-BPB mass than subword models. All panels are byte-weighted.

Table 15: Finite-range density fits $f_X \propto x^{\alpha}$ near zero and $f_X \propto e^{-x/\beta}$ in the tail, with β in BPB. Byte exponents are nearer -1; tail scales are comparable across granularities.

Model	Near-zero BPB	α	Tail BPB	β
Ours BYTE-1B	$[5.4 \times 10^{-5}, 2.15]$	-0.946	$[3, 15.5]$	1.898
BLT-1B	$[5.4 \times 10^{-5}, 2.86]$	-0.952	$[3.5, 15.5]$	1.855
H-Net 1-stage XL	$[4.1 \times 10^{-5}, 5.07]$	-0.916	$[5.5, 16]$	1.867
H-Net 2-stage XL	$[3.1 \times 10^{-5}, 5.07]$	-0.918	$[5.5, 16.5]$	1.899
Ours SUBWORD-1B	$[4.0 \times 10^{-4}, 0.164]$	-0.612	$[3.5, 13]$	1.684
Llama-3.1-8B	$[1.3 \times 10^{-5}, 0.164]$	-0.696	$[2.5, 11.25]$	1.407
Qwen3-8B	$[9.9 \times 10^{-7}, 0.0923]$	-0.789	$[3, 13.5]$	1.735
Qwen3.5-9B	$[3.1 \times 10^{-5}, 0.164]$	-0.691	$[2.5, 11.75]$	1.492

F.3 BPB AGGREGATION ABLATIONS

We test the sensitivity of BPB aggregation to tokenizer choice using BYTE-1B and BLT-1B. For each model, we average BPB over complete spans defined by the SUBWORD, Qwen3, and Qwen3.5 tokenizers and assign the span mean to each constituent byte. The aggregated distributions are similar across the three tokenizers (Figure 13).

Uniform four-byte grouping. To test whether average span length alone explains the aggregated distribution, we also use non-overlapping four-byte groups, close to SUBWORD's average token length in bytes. For both byte models, uniform grouping reduces variation but leaves more near-zero mass and less far-tail mass than tokenizer grouping (Figure 13). These differences highlight the role of tokenizer boundaries in shaping the aggregated BPB distribution.

Tokenizers share most boundaries. On the evaluation dataset, we find high pairwise boundary Jaccard similarity among the SUBWORD, Llama, Qwen3, and Qwen3.5 tokenizers (Figure 14). These similar boundaries group largely the same bytes together, helping explain the consistency of BPB aggregation across tokenizers.

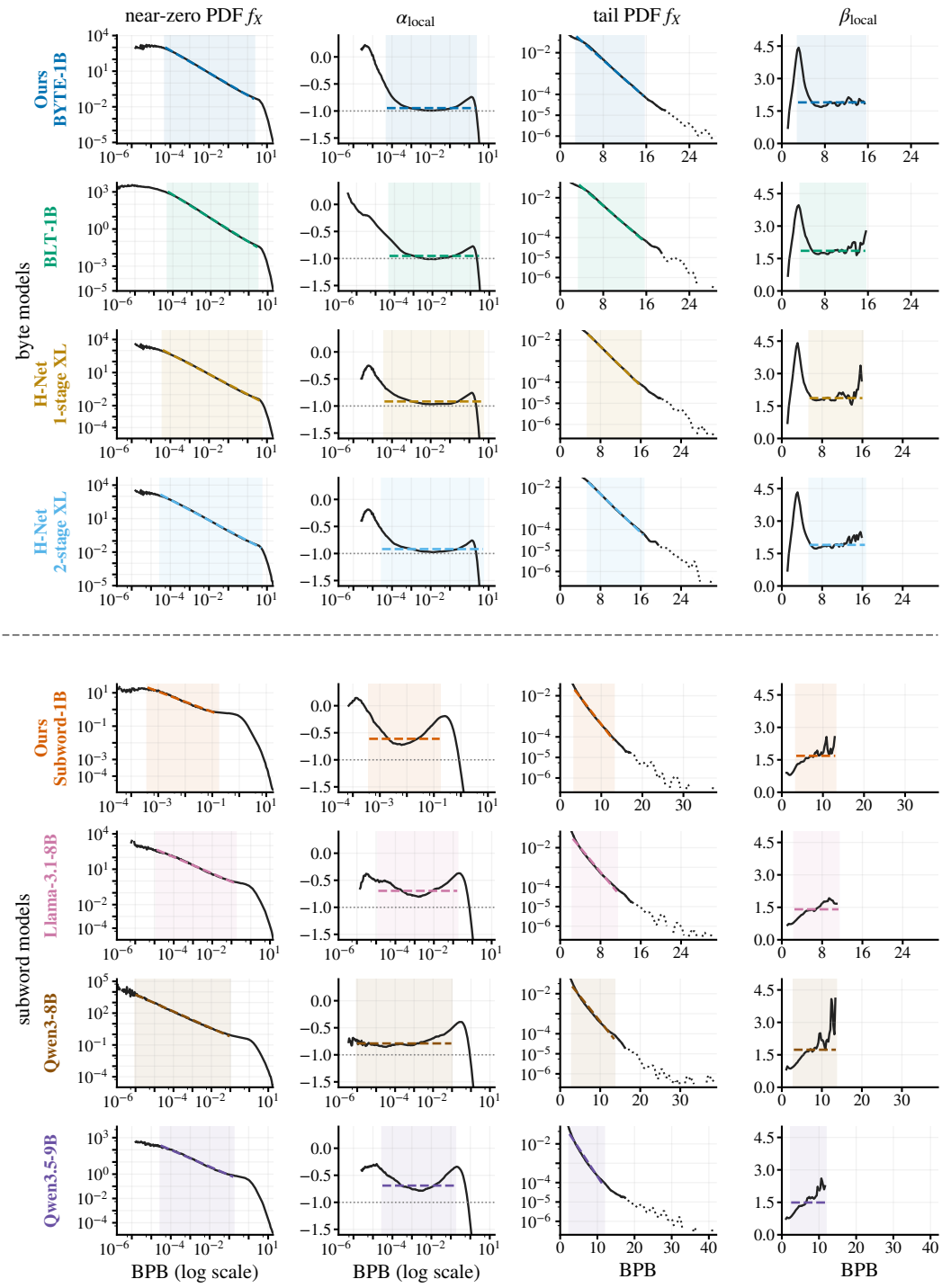

Figure 12: Near-zero power laws and exponential tail cores describe finite BPB ranges across models. All panels use byte-weighted distributions. Dashed lines show fits, and shading marks fitting ranges. Local exponents α_{local} and decay scales β_{local} indicate where the approximations hold. Dotted tail segments indicate sparse data. Fitting details are in Appendix F.2.

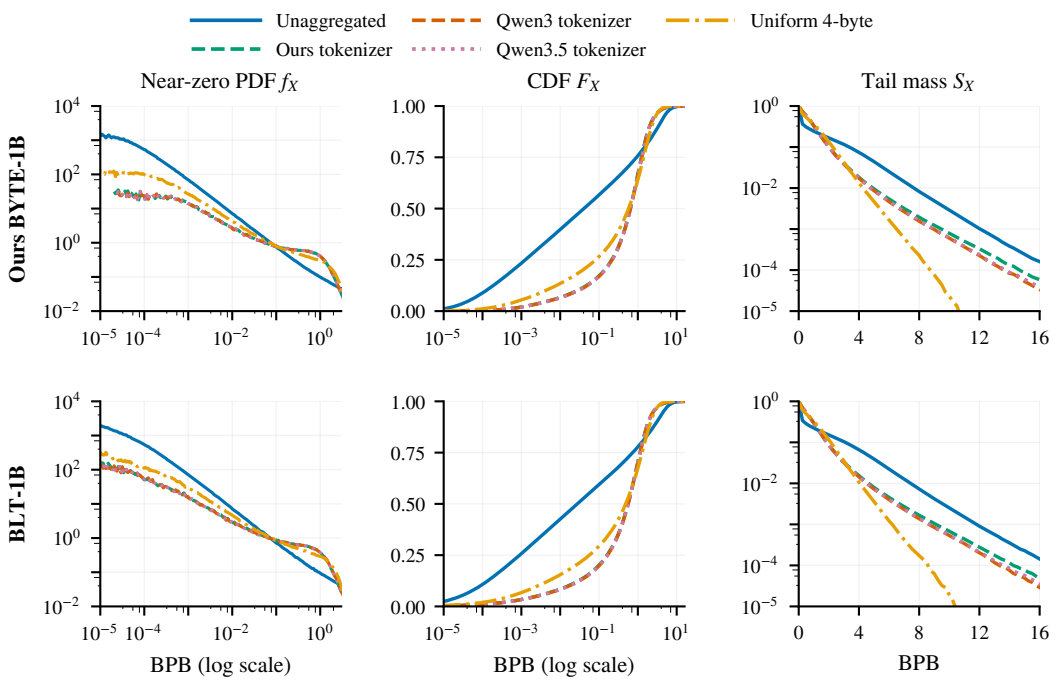

Figure 13: Token-span averaging reduces near-zero and high-BPB mass for both BYTE-1B and BLT-1B. All panels are byte-weighted. Tokenizers yield similar distributions; uniform 4-byte grouping leaves more near-zero mass and less far-tail mass (Appendix F.3).

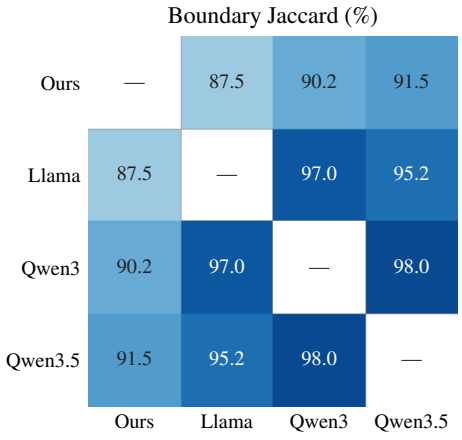

Figure 14: Tokenizers share most boundaries, consistent with their similar aggregation curves. Values are boundary Jaccard similarities (%).

G Draft Acceptance and Accepted Text

G.1 Models and evaluation settings

The 200M, 16-layer drafts and the 1B targets come from the main model suite, with the dense configurations in Table 5 and training settings in Appendix B. The 50M drafts use 8-layer models from the IsoFLOPs sweep.

We evaluate documents from the validation dataset in Appendix D.1 within each family's native context window, 8,192 byte tokens or 2,048 subword tokens, respectively.

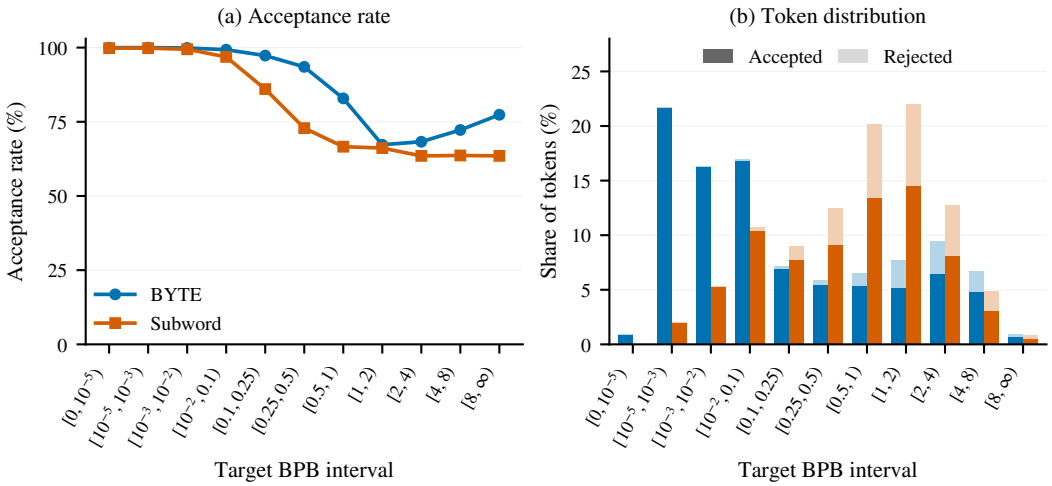

Figure 15: Small drafts achieve high acceptance rates on low-BPB byte tokens. Both families use 16-layer drafts and same-family 1B targets with verification on the given evaluation text within native windows. Left: acceptance rate within each target-BPB interval. Right: target-BPB distributions split into accepted (dark) and rejected (light) tokens, with each native token weighted equally. Colors identify model families.

G.2 Oracle simulation and acceptance metrics

To isolate draft–target agreement from the strategy used to construct draft candidates, we use an oracle simulation on the given evaluation text. It directly measures consecutive agreements, setting aside choices such as draft-tree expansion, pruning, and stopping heuristics.

Both models condition on the preceding tokens of this text at each scored token (teacher forcing). A draft token is counted as accepted when the two models select the same token, including when it differs from the reference token. The acceptance rate is the fraction of scored tokens accepted. The following step always uses the reference continuation as context. Each verification step ends at the first rejection or document boundary and counts as one simulated target forward pass, with no proposal-length cap.

The mean accepted draft length is the number of accepted draft tokens per simulated target forward pass. To compare text coverage across granularities, we also report accepted bytes per forward, summing the decoded byte lengths of the accepted draft tokens. Both averages include verification steps with no accepted tokens. Special tokens contribute no bytes; correction and bonus outputs are excluded.

Related work uses similar diagnostics of draft–target agreement. Mamou et al. (2024) let the draft continue until the first rejection, identifying the maximum accepted draft length at each iteration without choosing a fixed lookahead or a stopping heuristic. Their oracle follows generated histories, whereas ours measures agreement on the given evaluation text.

EvoESAP (Liu et al., 2026) likewise evaluates acceptance on fixed reference prefixes using teacher forcing. It averages the expected single-token acceptance computed from the overlap between draft and target next-token distributions. Our diagnostic measures consecutive agreements and their byte coverage.

Figure 15 shows high acceptance rates in the lowest-BPB intervals for both families. A much larger share of BYTE tokens lies in these intervals, and most of them are accepted. Together with the longer byte acceptance lengths in Table 3, this supports using small drafts to supply many low-BPB byte continuations for joint target verification.

Implications for speculative decoding. The drafting experiment provides a proof of concept for computation allocation at inference. Longer byte agreement runs indicate that small drafts can match

their targets across many consecutive byte tokens in reference text. These runs provide a teacher-forced diagnostic of drafting opportunities. Generated histories may differ from the reference text, so these lengths do not provide a general upper bound on accepted draft length during generation. In speculative decoding (Leviathan et al., 2023), accepted tokens become part of the subsequent context, and each round includes correction or bonus outputs. Establishing practical speedups therefore requires decoding along generated histories with finite proposals and measuring the costs of drafting and target verification.