B-NET: LEARNING WHERE TO COMPRESS IN BYTE-LEVEL LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hierarchical byte-level Transformers operate directly on bytes and group them into patches for efficient computation. Their patcher must identify useful boundaries without language-specific rules while keeping the resulting computational budget predictable. Space-based patchers encode language-specific structure, while learned patchers controlled by a soft ratio loss can deviate substantially from requested patch lengths. We introduce B-Net, a hierarchical byte-level Transformer that separates learned boundary placement from explicit compression control. A causal convolutional scorer learns input-dependent boundaries without a tokenizer or manually specified language features, while an adaptive threshold controller realizes the requested average patch rate. On English, the learned patcher recovers the quality of strong space-based segmentation without receiving spaces as boundary rules. The same patching mechanism remains effective in separately trained Chinese models. We compare B-Net with state-of-the-art byte-level Transformers under a FLOP- and data-matched protocol. B-Net outperforms the strongest byte-level baseline in each setting by 3.4 percentage points in mean downstream accuracy on English and 4.1 points in XWinograd accuracy on Chinese. It improves English WikiText performance from 0.85 to 0.81 bits per byte and matches the best Chinese validation bits per byte. These results show that learned byte patching can combine reliable rate control with superior predictive performance without manually specified language-boundary rules.

1 INTRODUCTION

Learning directly from bytes allows a language model to build representations of text without first committing to a subword vocabulary. Byte-level models (Xue et al., 2022) retain access to spelling and character structure, and the same input representation accommodates different languages without redesigning a tokenizer. These benefits come at a computational cost: byte sequences are longer than subword sequences, and applying a large network at every byte spends substantial computation on fine-grained predictions. Making byte-level language modeling competitive therefore requires a way to concentrate model capacity on useful groups of bytes.

Hierarchical byte-level Transformers (Nawrot et al., 2022; Yu et al., 2023; Pagnoni et al., 2025) provide this structure by grouping byte representations into patches and processing the shorter sequence with a larger inner network. However, shortening the sequence alone does not determine whether those patches are useful. Their boundaries determine how information is grouped, while their number determines how often the expensive inner network runs. A patcher must therefore solve two connected problems: learning where to place boundaries and controlling how much computation the resulting sequence requires.

Existing approaches offer different solutions to these problems. SpaceByte (Slagle, 2024) obtains strong English performance by aligning patches with predefined textual boundaries, while Byte Latent Transformer (BLT) (Pagnoni et al., 2025) uses predictions from a separate entropy model. H-Net (Hwang et al., 2026) learns boundaries jointly with the language model and encourages a requested patch length through a soft ratio loss. Yet learned boundary placement does not by itself ensure reliable rate control: in our seed study, H-Net's realized patch lengths remain substantially below the requested target. The challenge is to recover the predictive quality of useful segmentation rules without prescribing those rules, while making the average patch rate a dependable model setting.

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.

We introduce B-Net, a hierarchical byte-level Transformer that separates learned boundary placement from explicit rate control. Its patcher learns input-dependent boundaries without manually specified language rules, while an adaptive controller regulates the average patch rate.

Our contributions are:

1. **Reliable, adaptive byte patching.** B-Net tracks requested average patch lengths throughout training with little variation across seeds, while adapting individual boundaries to the input.
2. **Generality without language-specific rules.** The learned patcher recovers the predictive quality of strong space-based segmentation on English and remains effective in separately trained Chinese models, where spaces generally do not delimit words.
3. **Superior predictive performance.** Under a FLOP- and data-matched protocol, B-Net outperforms the strongest byte-level baseline in each setting by 3.4 percentage points in mean downstream accuracy on English and 4.1 points in XWinograd accuracy on Chinese. It also surpasses the byte-pair-encoding (BPE) reference on both English and Chinese evaluations.

To separate patching from backbone design, we introduce a modular framework covering Hourglass (Nawrot et al., 2022), MegaByte (Yu et al., 2023), Funnel-Transformer (Dai et al., 2020), SpaceByte (Slagle, 2024), AU-Net (Videau et al., 2025), Dynamic Pooling (Nawrot et al., 2023), H-Net (Hwang et al., 2026), and ByteFlow (Deng et al., 2026). It expresses the compared architectures through interchangeable components for patching, pooling, sequence processing, and reconstruction, enabling complete-model comparisons and patcher ablations within shared architectures.

2 Related Work

Tokenization and byte-level modeling. Byte-pair encoding (BPE) (Sennrich et al., 2016) learns a subword vocabulary before language-model training. ByT5 (Xue et al., 2022) models bytes directly, but the longer sequences increase computational cost. MambaByte (Wang et al., 2024) addresses this cost through selective state-space processing without patching, while Local Byte Fusion (Sreedhar et al., 2023) aggregates byte n-grams or word-delimited spans for machine translation. B-Net reduces the sequence length through variable-length patches learned jointly with the language model.

Hierarchical architectures and rule-based patching. Hourglass (Nawrot et al., 2022) and MegaByte (Yu et al., 2023) reduce autoregressive processing costs through fixed grouping, while Funnel-Transformer (Dai et al., 2020) provides an encoder-side precedent for sequence pooling. SpaceByte (Slagle, 2024) aligns variable-length patches with space-like textual boundaries, and AU-Net (Videau et al., 2025) combines text-boundary rules with word-count grouping. These approaches group positions at fixed intervals or according to supplied textual rules. B-Net retains the hierarchical processing structure while learning boundaries from the language-model objective.

Adaptive patching and rate control. Adaptive patchers differ in how they obtain boundary signals and regulate patch counts. Byte Latent Transformer (BLT) (Pagnoni et al., 2025) uses a separately trained entropy model and calibrates a threshold for a desired average patch size. Dynamic Pooling (Nawrot et al., 2023) explores both externally supervised boundaries and joint learning through a Gumbel-sigmoid estimator based on the Gumbel-softmax relaxation (Jang et al., 2017); Dauncey & Wattenhofer (2026) instead optimize discrete boundaries using score-function estimators. H-Net (Hwang et al., 2026) jointly learns cosine-similarity boundaries and encourages a target patch length through a soft ratio loss. Adaptive Targeted Dynamic Chunking (ATDC) (Dang et al., 2026) changes that target during training, including a loss-triggered adjustment, while retaining H-Net's routing rule and balancing loss. ByteFlow (Deng et al., 2026) instead constrains the selected count through full-sequence Top-K ranking of coding-rate scores during training. B-Net jointly learns boundary scores and adapts a separate selection threshold to regulate the average patch rate. At inference, this threshold is fixed, allowing prefix-based decisions and input-dependent patch counts.

Computational efficiency. Compute-optimal training studies (Hoffmann et al., 2022) show that allocating a fixed budget between model size and training tokens affects predictive quality. In byte-

level modeling, SpaceByte compares architectures under controlled compute budgets, while AU-Net studies equal-data budgets and scaling. Learned patching introduces an additional source of variation: the realized patch count determines how many positions the inner network processes. B-Net's explicit rate control addresses this variation, complementing architectural choices and training-budget allocation.

3 B-NET

Hierarchical byte-level Transformers. Byte-level Transformers predict text directly from bytes, avoiding a predefined subword tokenizer but processing longer sequences. Hierarchical architectures reduce this cost by grouping bytes into patches and processing the resulting shorter sequence with an inner network.

These architectures share a common structure. Boundary selection determines where patches end, and pooling forms one representation per patch. Sequence-processing modules transform the representations at each resolution; reconstruction maps patch representations back to byte resolution for prediction. We developed a modular framework that expresses this structure through interchangeable components, separating patching from backbone design. Prominent architectures, including Hourglass, MegaByte, Funnel-Transformer, SpaceByte, AU-Net, Dynamic Pooling, H-Net, and ByteFlow, can be expressed as special instances of this framework through their component choices. This allows us to compare complete models and study alternative patchers within shared architectures. Appendix H describes the framework and how the individual architectures map to its components.

B-Net overview. B-Net instantiates this framework with a learned patcher that separates boundary placement from compression control. It retains the hierarchical backbone of H-Net, with Mamba-2 (Dao & Gu, 2024) encoder and decoder stacks and a Transformer (Vaswani et al., 2017) inner stack, and replaces its patching mechanism with convolutional boundary scoring and an adaptive threshold controller. The scorer learns where to place boundaries, while the controller regulates how many positions reach the inner network.

Let $x_1, \ldots, x_S$ be an input sequence of S bytes, and let $X \in \mathbb{R}^{S \times D}$ contain its encoder representations, where D is the encoder width. The row $X_t \in \mathbb{R}^D$ represents position $t \in \{1, \ldots, S\}$ and depends only on bytes $x_1, \ldots, x_t$. The patcher selects positions as patch endpoints, which determine the representations supplied to the inner network. Let $n > 0$ denote the number of positions retained by the selection procedure actually used; the realized compression rate is S/n. We distinguish this realized rate from the requested average patch length $L_{\text{target}} \geq 1$. B-Net learns where to place boundaries while adapting a shared decision threshold to keep their count near that implied by the requested rate.

3.1 CONVOLUTIONAL BOUNDARY SCORING

The scorer assigns a boundary probability to each encoder representation. Let Conv_θ denote a one-dimensional causal convolution with learned parameters θ and positive integer window size w. Its output at position t is a scalar logit ℓ_t, computed from at most w consecutive input positions ending at t. Using the sigmoid function $\sigma(z) = 1/(1 + \exp(-z))$ for a scalar z, we obtain the boundary probability $p_t \in (0, 1)$:

$$\ell_t = \big(\mathrm{Conv}_\theta(X)\big)_t, \qquad p_t = \sigma(\ell_t). \tag{1}$$

The window size w controls local aggregation, while the controller determines patch lengths through its boundary decisions.

3.2 ADAPTIVE THRESHOLD CONTROL

We select a boundary when its predicted probability exceeds a shared threshold. Raising the threshold selects fewer boundaries and produces longer patches; lowering it selects more boundaries and produces shorter patches. For training, index sequences by i and optimizer steps by u. Sequence i has length S_i and boundary probabilities $p_{i,t}$ for positions $t \in \{1, \ldots, S_i\}$; we suppress the step index on these quantities. Let $b_{i,t} \in \{0, 1\}$ denote its raw threshold decisions, before the warmup

selection or capacity safeguard described below. Let τ_u be the threshold held fixed throughout step u, and let n_i^{raw} count the selected positions before any capacity correction. Writing $\mathbf{1}[A]$ for the indicator that equals one when condition A holds and zero otherwise, we define

$$b_{i,t} = \mathbf{1}[p_{i,t} > \tau_u], \qquad n_i^{\text{raw}} = \sum_{t=1}^{S_i} b_{i,t}, \tag{2}$$

After warmup, these raw decisions determine the retained boundaries unless the capacity safeguard removes excess positions. As boundary probabilities change during training, the same threshold can select different numbers of boundaries. For sequence i, let n_i^{target} be the integer target patch count corresponding to L_{target}, with $n_i^{\text{target}} \approx S_i / L_{\text{target}}$. We therefore adapt the threshold to keep the selected boundary count close to this target. The target-count ranking below estimates the score at which to place this threshold. At step u, each sequence i with a positive target supplies a cutoff estimate $q_{u,i}$. Let TopK_k return the k largest values of its input, where k is a positive integer no larger than the number of input values. We compute the cutoff estimate by taking the sequence's n_i^{target} highest boundary probabilities and choosing the smallest among them:

$$q_{u,i} = \min\Big(\text{TopK}_{n_i^{\text{target}}}(p_{i,1}, \ldots, p_{i,S_i})\Big), \tag{3}$$

Let $\mathcal{B}_u$ be the set of indices of all such sequences processed during step u, with $|\mathcal{B}_u|$ denoting their number. If this set is empty, the threshold remains unchanged; otherwise, we update it as follows. We average these per-sequence estimates to obtain $\bar{q}_u$, a single cutoff estimate for the step. After the optimizer step completes, the controller moves the shared threshold τ_u toward this mean using an exponential moving average (EMA) with decay $\beta \in [0, 1)$:

$$\begin{aligned} \bar{q}_u &= \frac{1}{|\mathcal{B}_u|} \sum_{i \in \mathcal{B}_u} q_{u,i}, \\ \tau_{u+1} &= \beta \tau_u + (1 - \beta) \bar{q}_u, \end{aligned} \tag{4}$$

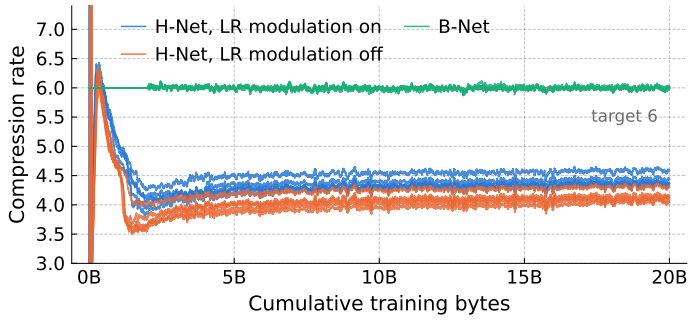
The delay ensures that current-step scores affect only subsequent thresholds. The threshold EMA uses detached scores and provides no gradient to the boundary scorer. Because averaging cutoffs does not enforce an exact boundary count, we assess target tracking empirically in Section 4.2.

Threshold warmup. The initial threshold may select a boundary count far from the target, so we calibrate it before using it for selection. Let W denote the number of warmup optimizer steps. During these steps, we select the n_i^{target} highest-scoring positions in each sequence and update the threshold EMA after each optimizer step. We then switch to threshold-based selection and continue adapting the threshold throughout training. If the selected count exceeds a configured memory cap, a training-only safeguard retains the highest-scoring selected boundaries up to that cap; otherwise, it leaves the selection unchanged. Appendix A gives the implementation details and measured clipping frequencies.

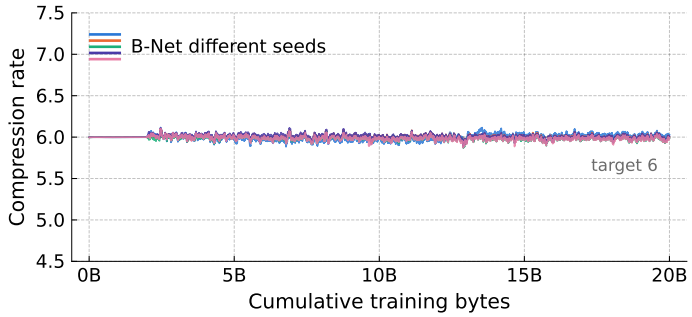
Inference and causality. At inference, we freeze the threshold learned during training and select boundaries directly from the causal convolutional scores, without ranking or capacity clipping. Each boundary decision therefore depends only on the observed prefix. For the complete model to remain causal, inner processing must also be causal, and reconstruction may use a completed patch representation only at or after its endpoint. During training, warmup and capacity clipping rank scores across the sequence, so later positions can affect earlier boundary decisions in those operations. The delayed threshold update does not remove this dependence through ranking; causal inference therefore does not imply strictly causal training. The frozen inference threshold also allows realized compression to differ from the training target. Appendix D details the causality requirements for all components.

3.3 Learning the Boundary Scorer

Hard boundary decisions do not propagate gradients to the boundary scorer, so we retain the selected sigmoid probabilities for differentiable reconstruction. This feature smoothing carries gradients from the prediction loss; the threshold EMA above only updates the rate controller's state. For



(a) H-Net and B-Net, target 6



(b) B-Net seeds, target 6

Figure 1: **Patch-length histories across seeds at target six.** (a) Ten completed H-Net runs, five seeds with layer-wise LR modulation on (blue) and off (orange), alongside B-Net (green), plotted against training bytes. (b) Five completed B-Net runs across the full 20B-byte training budget, showing fluctuations around six after warmup. B-Net and both H-Net settings use the same five seeds. Curves use a trailing mean over 100 logged readings.

a single sequence, index retained patches by $j \in \{1, \ldots, n\}$ and let $e_1 < \cdots < e_n$ denote their selected endpoints in sequence order, after any warmup selection or capacity correction. Pooling selects X_{e_j} as the patch's input representation. Let $D_{\text{inner}} \geq D$ be the inner Transformer's width. We append learned features to expand each pooled representation from D to D_{inner} dimensions before inner processing, then retain the first D output features for reconstruction. Let $Z_j \in \mathbb{R}^D$ denote the resulting representation of patch j. Following the EMA smoothing of H-Net (Hwang et al., 2026), we combine Z_j with the preceding smoothed representation, using the boundary probability p_{e_j} as the interpolation weight. Writing $\widetilde{Z}_j$ for the resulting smoothed representation and initializing $\widetilde{Z}_0$ to the zero vector, we compute

$$\widetilde{Z}_j = p_{e_j} Z_j + (1 - p_{e_j}) \widetilde{Z}_{j-1}, \qquad \widetilde{Z}_0 = \mathbf{0}. \tag{5}$$

Copy upsampling restores byte resolution by reusing the most recent causally available smoothed patch representation at each position. The decoder's next-byte cross-entropy loss thus trains the boundary scorer through the selected probabilities while treating the patch indices as fixed. Our reference configuration uses this gradient path without an auxiliary ratio loss or an upsampling straight-through estimator (Bengio et al., 2013).

4 Experiments

4.1 Experimental Setup

We evaluate rate-control reliability, predictive performance, and the generality of learned patching across English and Chinese. We follow the English and Chinese comparison setting of H-Net, using FineWeb-Edu (Penedo et al., 2024) and FineWeb-Edu-Chinese-V2.1 (Yu et al., 2025). We train large English models on 150B bytes and small English and Chinese models on 20B bytes.

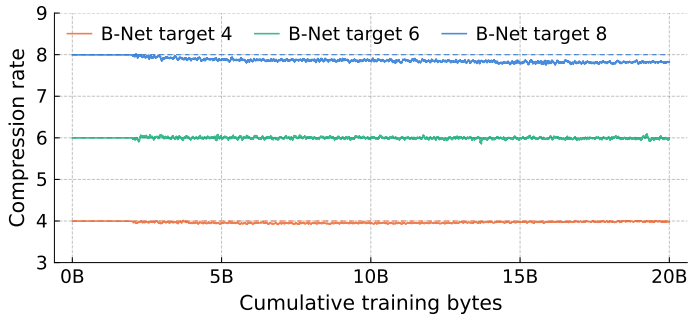


Figure 2: **B-Net tracks different requested patch lengths throughout training.** Three small-English runs use the same model configuration with targets four (orange), six (green), and eight (blue); dashed lines mark the requested rates. Each history spans 20B training bytes. Over the final 0.2B bytes, the runs reach average compression rates of 3.99, 5.98, and 7.82, respectively. Curves use a trailing mean over 100 logged readings.

Byte models use 8192-byte contexts; BPE references use 1792-token contexts with GPT-2 (Radford et al., 2019) tokenization for English and Qwen2.5 (Qwen et al., 2025) for Chinese. Appendix H.3 records the model configurations. All code for our implementation and experiments is provided in an anonymous Git repository at `https://anonymous.4open.science/r/modular-transformer-anonymous-2E90`.

Baselines and matching protocol. We compare BPE and direct byte Transformers, fixed-patch models, SpaceByte, AU-Net, Dynamic Pooling, H-Net, and ByteFlow in a common modular implementation. We omit BLT because we could not establish a reliable reproduction with the released implementation; Appendix H discusses this limitation. Appendix G documents our ByteFlow implementation. We follow H-Net's FLOP-counting protocol, matching models by analytical forward FLOPs per byte rather than measured cumulative training FLOPs. Within each setting, we match training corpora and byte budgets and adjust depth to approximately match the BPE reference's forward FLOPs per byte; parameter counts may differ. English FLOP estimates deviate by at most 7.6% at large scale and 8.1% at small scale. Appendix I details the FLOP calculations, counting conventions, and limitations.

4.2 Reliability: Requested and Realized Patch Rates

We test whether a requested patch length reliably determines the compressed sequence length throughout training and across seeds. Figure 1(a) shows ten completed small-scale H-Net runs at target six, using five seeds with and without H-Net's layer-wise learning-rate modulation (Appendix B). Their mean compression rates over the final 0.2B training bytes range from 4.03 to 4.57. On the FineWeb-Edu test set, the across-seed means and standard deviations of the compression rate are 4.394 ± 0.103 with modulation and 4.121 ± 0.099 without it. Neither learning-rate setting makes H-Net's nominal target a reliable predictor of realized compression in this study.

For a fixed input batch, a realized patch length of 4.5 instead of six produces one-third more inner positions and increases the inner network's activation memory requirements. A batch sized for the requested target can therefore exceed GPU capacity as H-Net's patch lengths decrease during training.

Sustained target tracking across seeds. Across five completed runs, B-Net tracks six throughout the 20B-byte training budget, including after warmup, with small fluctuations (Figure 1(b)). On the same FineWeb-Edu test set, B-Net achieves a compression rate of 5.986 ± 0.015 (across-seed mean and standard deviation), with a mean 0.23% below the target. Together, the trajectories and test-set measurements show that B-Net maintains the requested compression during training and retains it on held-out data across seeds.

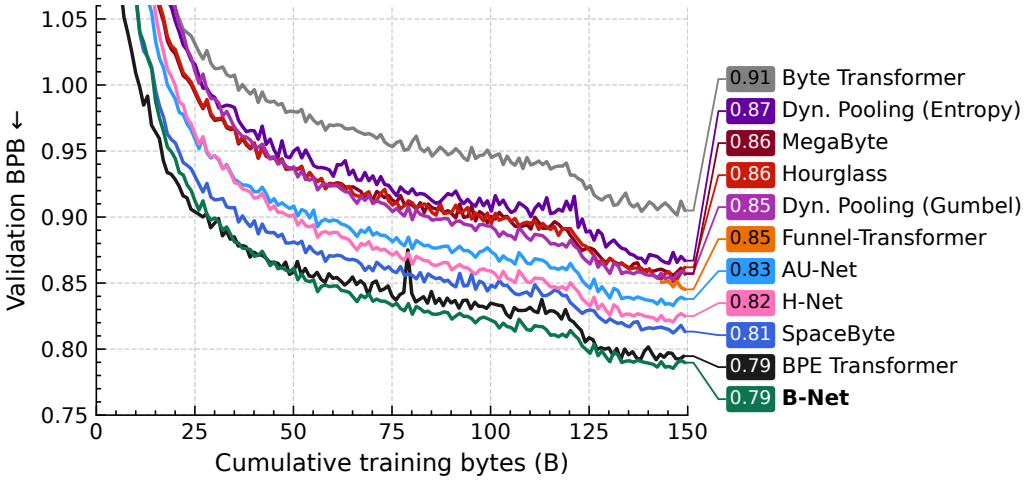


Figure 3: **Validation BPB of large models on FineWeb-Edu.** Curves show unsmoothed intermediate validation measurements against cumulative training bytes; lower is better. Line hues group models by patching mechanism and match the row shading in Table 1. Boxed values give each model's final BPB over the full validation set, and labels are ranked by the unrounded values.

Tracking different requested rates. We test control beyond target six by requesting four, six, and eight with the same architecture and training recipe (Figure 2). Mean patch lengths over the final 0.2B bytes are 3.99, 5.98, and 7.82, respectively, with a small downward offset at target eight. The trajectories show sustained tracking across the tested compression range. All runs use 20B training bytes; computation differs across targets.

Training capacity safeguard. We quantify reliance on the memory safeguard by measuring how often it overrides threshold selection. After warmup, B-Net's safeguard activates for an estimated 0.157% of large-English sequences and 0.90% of small-English sequences. H-Net exceeds the same capacity cap on at least 7.277% and 12.264% of sequences, respectively, without applying clipping. A separate Chinese diagnostic logs a B-Net clipping rate of 6.701%. Corrections remove 0.0162%, 0.1415%, and 1.7286% of proposed boundaries in the three settings, respectively; Appendix A.1 provides the full measurements. Threshold selection therefore operates without correction on most sequences, although the safeguard is more active in the Chinese diagnostic. Clipping is disabled at inference, so corrected training counts do not establish frozen-threshold inference rates.

4.3 Performance: Complete-Model Comparisons

We test predictive quality under the FLOP- and data-matching protocol of Section 4.1.

We report predictive loss in bits per byte (bpb), dividing summed natural-log cross-entropy by the number of scored bytes and by $\ln 2$. English zero-shot evaluation uses the Language Model Evaluation Harness (Gao et al., 2024); the tables give task-level scores and their unweighted mean. We evaluate English predictive loss on WikiText (Merity et al., 2017) and downstream accuracy on LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), and XWinograd-en (Tikhonov & Ryabinin, 2021; Muennighoff et al., 2023). The large-scale mean includes these six downstream tasks and the small-scale mean additionally includes WinoGrande (Sakaguchi et al., 2021), so aggregate scores are comparable only within each setting. HellaSwag and ARC-Challenge use length-normalized accuracy (acc_n); the other downstream tasks use accuracy (acc). The performance tables report point estimates of predictive loss and task accuracy.

At the larger English model scale, B-Net achieves the lowest WikiText loss (0.81 bpb) and highest mean downstream accuracy (56.2%) in Table 1. It outperforms SpaceByte, the strongest byte-level baseline at 52.8%, by 3.4 percentage points and exceeds the BPE reference at 55.4%. H-Net uses the

same backbone but reaches 0.85 bpb and 51.5% accuracy, a gap of 4.7 points that supports B-Net's patch quality alongside its rate control. BPE remains ahead on PIQA and both ARC tasks; B-Net's advantage is in aggregate performance.

Table 1: **Zero-shot evaluation of large models across multiple benchmarks.** Models use the same training-data budget and approximately matched forward-compute estimates; see Section 4.1. Accuracy is reported in percent. Mean averages LAMBADA, HellaSwag, PIQA, ARC-Easy, ARC-Challenge, XWinograd-en, and WinoGrande. Wiki denotes WikiText. LAMBADA uses the OpenAI variant; XW-en is the English XWinograd subset. Random denotes uniform choice selection, with dashes for WikiText, LAMBADA, and the seven-task Mean. Best results are bold and second-best results are underlined, ranked before rounding.

Model	Wiki bpb ↓	LMB. acc ↑	Hella. acc_n ↑	PIQA acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	XW-en acc ↑	Wino. acc ↑	Mean acc ↑
Random	–	–	25.0	50.0	25.0	25.0	50.0	50.0	–
BPE Transformer	<u>0.82</u>	40.2	<u>50.0</u>	**70.5**	**67.1**	**33.4**	<u>71.4</u>	**55.4**	<u>55.4</u>
Byte Transformer	0.96	33.3	37.1	63.7	54.8	24.8	61.1	50.9	46.5
MegaByte	0.90	35.0	42.5	64.6	55.8	29.2	64.9	50.7	49.0
Hourglass	0.89	35.9	42.7	67.2	56.9	27.1	63.4	51.0	49.2
Funnel-Transformer	0.88	35.2	43.4	66.4	57.2	29.3	64.0	49.3	49.3
SpaceByte	0.85	<u>40.9</u>	46.7	68.3	62.0	31.0	67.7	52.4	52.7
AU-Net	0.86	37.2	43.4	67.0	58.6	27.6	67.2	51.4	50.4
Dyn. Pooling (Entropy)	0.92	32.9	42.6	64.7	57.3	30.2	64.3	50.0	48.8
Dyn. Pooling (Gumbel)	0.89	33.7	42.7	64.4	54.8	27.5	66.6	50.1	48.5
H-Net	0.85	36.4	46.3	68.0	60.3	30.0	68.1	52.6	51.7
ByteFlow	1.05	5.6	28.7	53.2	31.9	21.5	52.6	50.5	34.9
B-Net	**0.81**	**43.0**	**51.9**	<u>70.0</u>	<u>65.9</u>	<u>32.2</u>	**74.1**	<u>55.4</u>	**56.1**

To assess performance beyond standard input formatting, we evaluate five HellaSwag perturbations (Appendix E.1). B-Net leads on their mean accuracy, 34.0% versus 33.3% for SpaceByte, extending its aggregate advantage to perturbed text without winning every condition.

Learning curves and model scale. Figure 3 tests whether the checkpoint gains are also reflected in validation loss during training. B-Net remains among the lowest-loss models over much of training and finishes below the other byte-level models, supporting its advantage beyond downstream task accuracy. The horizontal axis measures training bytes rather than accumulated computation.

We also test smaller English models with less training data (Appendix E). B-Net again leads in mean accuracy at 44.6%, compared with 44.4% for BPE, 42.8% for SpaceByte, and 42.5% for H-Net. Its advantage thus appears at both tested scales.

Component evidence. The component ablations test which patching choices support B-Net's predictive quality (Appendix F). Replacing adaptive threshold control with a ratio loss reduces small English mean accuracy from 44.6% to 42.7%; reducing the convolution kernel width from four to one yields 43.0%, and substituting cosine-similarity scoring also yields 43.0%. The space-like variant reaches 44.5%, showing that learned patching recovers the English heuristic's quality without receiving its boundary rule. The component comparisons are FLOP- and data-matched. Because B-Net retains H-Net's backbone, the gains demonstrate the benefit of replacing its patching mechanism with our learned scorer and adaptive rate controller.

4.4 Generality: Chinese Evaluation

Chinese tests whether B-Net's predictive advantage extends to a language where spaces generally do not mark word boundaries. We train Chinese models separately with the same B-Net patching mechanism and a target compression rate of 3.5. On XWinograd-zh (Table 2), B-Net reaches 58.7%,

outperforming the strongest byte-level baseline, Dynamic Pooling (Gumbel) at 54.6%, by 4.1 percentage points. SpaceByte reaches 52.6%, H-Net 53.4%, and BPE 56.0%. B-Net has the lowest unrounded validation bpb, although both B-Net and H-Net round to 0.98. Together with the English results, these separately trained models show that the same patching design achieves superior predictive performance in both languages without language-specific boundary rules.

Table 2: **Chinese validation and XWinograd results.** Models train on 20B bytes of FineWeb-Edu-Chinese-V2.1. XWinograd-zh accuracy is in percent; Random denotes uniform choice and has no bpb score. Best scores are bold and second-best scores are underlined, ranked before rounding.

Model	Validation bpb ↓	XWinograd-zh acc ↑
Random	–	50.0
BPE Transformer	0.99	<u>56.0</u>
Byte Transformer	1.11	52.8
MegaByte	1.09	53.8
Hourglass	1.10	51.6
Funnel-Transformer	1.06	52.8
SpaceByte	1.05	52.6
AU-Net	1.07	54.4
Dyn. Pooling (Entropy)	1.08	54.0
Dyn. Pooling (Gumbel)	1.06	54.6
H-Net	<u>0.98</u>	53.4
ByteFlow	1.12	48.6
B-Net	**0.98**	**58.7**

5 Limitations and Conclusion

Limitations. Our experiments cover English and Chinese at the reported model scales, so they do not establish performance across arbitrary languages, non-text data, or deeper hierarchies. Inference uses prefix-causal scoring and a frozen threshold, but sequence-wide ranking during warmup and capacity clipping makes the training procedure different from strictly prefix-causal inference. The frozen threshold also permits inference compression to deviate from the training target under distribution shift. These limits make held-out rate measurements and inference causality checks necessary alongside predictive quality.

Conclusion. Efficient byte-level language modeling calls for patching that preserves predictive quality without language-specific boundary rules and keeps computation predictable. We introduced B-Net to address these requirements by separating learned boundary placement from explicit control of the patch rate. The repeated H-Net runs demonstrate that a nominal target alone does not ensure the requested compression. B-Net maintains close target tracking throughout training across the tested seeds and rates, with little across-seed variation in held-out compression.

On English, learned patching recovers the quality of space-based segmentation without language-specific rules; the same mechanism remains effective in separately trained Chinese models. With H-Net's backbone retained, the gains in bits per byte and downstream accuracy demonstrate the benefit of B-Net's learned scorer and adaptive rate controller. Under our FLOP- and data-matched protocol, B-Net outperforms the strongest byte-level baseline and the BPE reference on aggregate English accuracy and Chinese XWinograd accuracy. Component ablations within our modular framework support this combination of reliable compression control, useful learned boundaries, and superior predictive performance.

AI USE STATEMENT

We used generative AI tools to assist with drafting and polishing sections of the manuscript, finding related work, and writing code for the research implementation. The authors take responsibility for the final manuscript, code, and reported results.

REPRODUCIBILITY STATEMENT

To support reproduction of our results, we provide all code for our implementation and experiments through an anonymous Git repository at `https://anonymous.4open.science/r/modular-transformer-anonymous-2E90`. Section 3 specifies boundary scoring, adaptive rate control, and the reconstruction mechanism used to train the scorer. Section 4.1 describes the datasets, training budgets, and baselines; Section 4.3 defines the predictive-performance metrics. The appendices document the threshold updates and training safeguards (Appendix A), autoregressive causality (Appendix D), architecture components and model configurations (Appendix H), and the FLOP-counting methodology and its limitations (Appendix I).

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *AAAI*, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv:1803.05457*, 2018.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. Funnel-Transformer: Filtering out sequential redundancy for efficient language processing. In *NeurIPS*, 2020.

Thang Dang, Akira Nakagawa, Kenichi Kobayashi, and Koichi Shirahata. Adaptive targeted dynamic chunking for tokenization-free hierarchical model. *arXiv:2605.30080*, 2026.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *ICML*, 2024.

Sam Dauncey and Roger Wattenhofer. You can learn tokenization end-to-end with reinforcement learning. In *ICML*, 2026.

Chunyuan Deng, Sanket Lokegaonkar, Colin Lockard, Besnik Fetahu, Nasser Zalmout, and Xian Li. ByteFlow: Language modeling through adaptive byte compression without a tokenizer. In *ICLR*, 2026.

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

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *NeurIPS*, 2022.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. In *ICLR*, 2026.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *ICLR*, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. Crosslingual generalization through multitask finetuning. In *ACL*, 2023.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In *Findings of NAACL*, 2022.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In *ACL*, 2023.

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 *ACL*, 2025.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*, 2016.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *NeurIPS*, 2024.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv:2412.15115*, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 2021.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

Noam Shazeer. GLU variants improve transformer. *arXiv:2002.05202*, 2020.

Kevin Slagle. SpaceByte: Towards deleting tokenization from large language modeling. In *NeurIPS*, 2024.

Makesh Narsimhan Sreedhar, Xiangpeng Wan, Yu Cheng, and Junjie Hu. Local byte fusion for neural machine translation. In *ACL*, 2023.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Alexey Tikhonov and Max Ryabinin. It's All in the Heads: Using Attention Heads as a Baseline for Cross-Lingual Transfer in Commonsense Reasoning. In *Findings of ACL-IJCNLP*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Mathurin Videau, Badr Youbi Idrissi, Alessandro Leite, Marc Schoenauer, Olivier Teytaud, and David Lopez-Paz. From bytes to ideas: Language modeling with autoregressive U-Nets. In *NeurIPS*, 2025.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. MambaByte: Token-free selective state space model. In *COLM*, 2024.

Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

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. *TACL*, 2022.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *NeurIPS*, 2023.

Yijiong Yu, Ziyun Dai, Zekun Wang, Wei Wang, Ran Chen, and Ji Pei. OpenCSG chinese corpus: A series of high-quality chinese datasets for LLM training. *arXiv:2501.08197*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *ACL*, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *NeurIPS*, 2019.

APPENDIX OVERVIEW

The appendix collects supporting results, implementation details, and the compute-counting protocol.

Section	Contents	Page
A	Adaptive threshold implementation and clipping diagnostics	13
B	Layer-wise learning-rate modulation	14
C	Qualitative patch-boundary examples	14
D	Autoregressive causality	16
E	Additional evaluations, textual perturbations, generation examples, and learning curves	17
F	Component ablations	17
G	ByteFlow implementation and diagnostics	19
H	Architecture operations, component mapping, and model configurations	22
I	FLOP-counting methodology	27

A ADAPTIVE THRESHOLD IMPLEMENTATION DETAILS

Experimental settings. The three capacity-diagnostic runs in Table 3 use maximum-only clipping with tolerance $c = 0.25$. Their warmup lasts 3578 completed optimizer steps for large English and 3816 steps for small English and the Chinese diagnostic run, corresponding to approximately the first 10% of training bytes. The threshold EMA decay is $\beta = 0.999$; Table 10 collects the training and controller settings. The threshold starts at $\tau_0 = 0$ and uses no EMA bias correction, so its initial value need not yield the target boundary count. Top-K warmup therefore controls the count while the EMA accumulates cutoff estimates.

Integer target counts. For a nonempty sequence with S_i valid input positions, the architecture sets the target patch count to $n_i^{\text{target}} = \max(1, \lfloor S_i / L_{\text{target}} \rfloor)$, where $\lfloor \cdot \rfloor$ denotes rounding down to the nearest integer. The count uses the valid sequence length rather than its padded width. This floor rounding accounts for the approximation in the main-text relation $n_i^{\text{target}} \approx S_i / L_{\text{target}}$.

Patch-count clipping as a training safeguard. After Top-K warmup, the adaptive threshold selects boundaries according to Equation 2. Changes in boundary scores can nevertheless produce too many patches for GPU memory. We therefore use maximum-only clipping as a training safety net: if a sequence exceeds a configured patch-count cap, we retain only its highest-scoring boundaries up to that cap. For a relative tolerance $c \in [0, 1)$, the resulting count is

$$n_i^{\text{train}} = \min\left(n_i^{\text{raw}}, \left\lfloor (1 + c) n_i^{\text{target}} \right\rfloor\right). \tag{6}$$

Removing excess boundaries merges adjacent patches, limiting the length of the compressed sequence to prevent out-of-memory failures. This safeguard leaves threshold decisions unchanged for sequences below the cap, including those with fewer boundaries than the target. We disable this safeguard at inference and use the frozen threshold directly.

Causality of training selection. The convolution scores and frozen-threshold decisions depend only on the observed prefix. Warmup selection and patch-count clipping rank scores across a sequence, however, so later positions can affect earlier boundary decisions during training. Delaying the threshold update does not remove this dependence through ranking. We therefore distinguish these training operations from causal inference under the requirements in Appendix D.

Cutoff accumulation. Ranking and candidate-cutoff computation use detached scores converted to float32. Each training selection call accumulates the sum of its candidate cutoffs and the number of groups with a positive target count. After an optimizer step, these sums and counts are summed across data-parallel processes before their ratio is used in Equation 4. Thus, gradient accumulation and distributed training produce one controller update per completed optimizer step, rather than an update per microbatch or process.

A.1 Training Capacity Diagnostics

Table 3 distinguishes the fraction of sequences requiring clipping from the fraction of proposed boundaries removed. Statistics cover 8192-byte sequences after warmup, with a cap 25% above the target boundary count. B-Net rates measure actual clipping; the H-Net row gives lower bounds on the fraction of sequences exceeding the same cap, without applying clipping.

English clipping rates are estimated from logged microbatches, with 95% intervals over optimizer steps; the small-English interval also covers ambiguity in the logged microbatch counts. The Chinese rates are logged exactly in a separate diagnostic run, not the checkpoint evaluated in Table 2. The safeguard is used more frequently in this Chinese run than in either English setting.

Table 3: **Clipping frequency and boundary removal after warmup at a 25% capacity tolerance.**

Metric (%)	Large English	Small English	Chinese
B-Net: sequences clipped	0.157 ± 0.026	0.90 ± 0.13	6.701
B-Net: boundaries removed	0.0162	0.1415	1.7286
H-Net: sequences exceeding cap	≥ 7.277	≥ 12.264	≥ 9.127

B Layer-Wise Learning-Rate Modulation

H-Net multiplies the base learning rate of each hierarchy stage by a stage-specific factor, giving outer stages, which process longer sequences, larger learning rates than inner ones (Hwang et al., 2026, Section 2.3 and Appendix C). The outer stage operates on bytes, and the inner stage operates on the shorter sequence of patches formed by downsampling, which groups contiguous bytes. The factor grows with the number of positions a stage processes and shrinks with its width. H-Net motivates this modulation by more stable training and faster learning of boundary placement, since the outer stage places the boundaries that inner stages depend on. H-Net's rule gives

$$\lambda_{\text{outer}} = \sqrt{N_{\text{GPT}} \, \frac{D_{\text{inner}}}{D}}, \qquad \lambda_{\text{inner}} = \sqrt{\frac{N_{\text{GPT}}}{L_{\text{target}}}}, \tag{7}$$

where $N_{\text{GPT}} = 4.6$ is the average number of bytes per GPT-2 token on FineWeb-Edu. The inner factor applies to the inner Transformer, including its dimension expansion, and the outer factor applies to all other parameters, including the boundary scorer. In the small English seed study of Section 4.2, $D = 576$, $D_{\text{inner}} = 768$, and $L_{\text{target}} = 6$ give $\lambda_{\text{outer}} \approx 2.48$ and $\lambda_{\text{inner}} \approx 0.88$, or peak learning rates of 3.7×10^{-3} and 1.3×10^{-3}. With modulation disabled, all parameters use the base peak rate of 1.5×10^{-3}.

C Qualitative patch boundaries

Figures 4 and 5 show where each hierarchical model from the large English and small Chinese comparisons ends its patches on one sentence. Each model reads the same bytes in one seeded forward pass, so the Gumbel variant of Dynamic Pooling, which samples its boundaries, contributes a single draw. One sentence illustrates how each boundary rule behaves, but it cannot measure how often a model's patches align with words or characters. We therefore quote counts only for these sentences.

English. The fixed-stride models ignore the text. MegaByte and Hourglass produce identical eight-byte patches that cut through three words and split the emoji after its first byte, and Funnel-Transformer's six-byte patches cut through six words. SpaceByte and AU-Net place boundaries by rule at spaces, so each of their patches holds one word; AU-Net also separates the emoji from its following space. B-Net learns the same segmentation without such a rule: its seven boundaries all fall on spaces, and its patches match SpaceByte's exactly. The other learned patchers cut inside words, H-Net at 12 of its 13 boundaries and the entropy and Gumbel variants of Dynamic Pooling at 5 of 8 and 6 of 8, respectively. H-Net also forms the most patches (14), consistent with its shorter

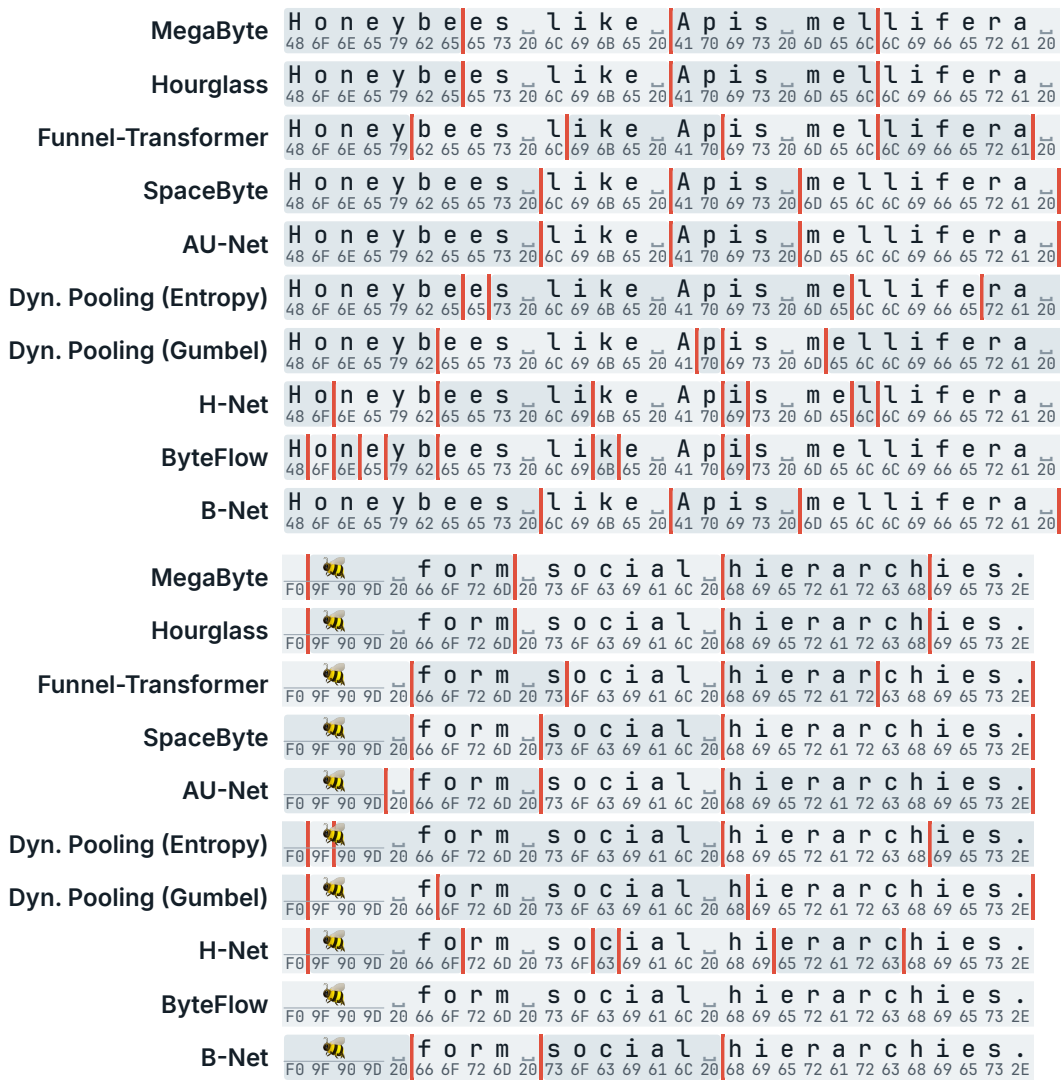


Figure 4: **Patch boundaries of the large English hierarchical models on one sentence.** Rows follow Table 11; the BPE and byte Transformers have no patcher and are omitted. Each row shows the characters above their UTF-8 bytes in hexadecimal; a small open box marks a space, and a gray line under the emoji spans its four bytes. Shaded boxes in alternating tones are patches, and a red bar follows each byte that the model selects as a patch boundary. A box cut flat at a line edge continues on the next line or past the displayed text.

realized patch lengths in Section 4.2. ByteFlow places all nine of its boundaries within the first 18 bytes and leaves the remaining 41 bytes in one patch, the concentration discussed in Appendix G.

Chinese. The three fixed-stride models use four-byte patches in Chinese and again produce identical segmentations that ignore character boundaries. SpaceByte's rule counts the first byte of each multi-byte character as space-like, so it ends a patch one byte into nearly every Chinese character. AU-Net splits only at spaces here, so the 45 bytes of Chinese text form a single patch. B-Net forms 20 patches, or 3.45 bytes per patch against its target of 3.5. Ten of its 12 boundaries in the Chinese text coincide with SpaceByte's, one byte into a character, and it skips four of SpaceByte's boundaries, so some of its patches span two characters. H-Net and the Gumbel variant of Dynamic Pooling form 23 patches each and cut characters at varying offsets: 9 of H-Net's boundaries fall after a character's first byte and 3 after its second, compared with 6 and 8 for the Gumbel variant. The entropy variant forms only 9 patches. ByteFlow forms 20 patches, as many as B-Net, but places

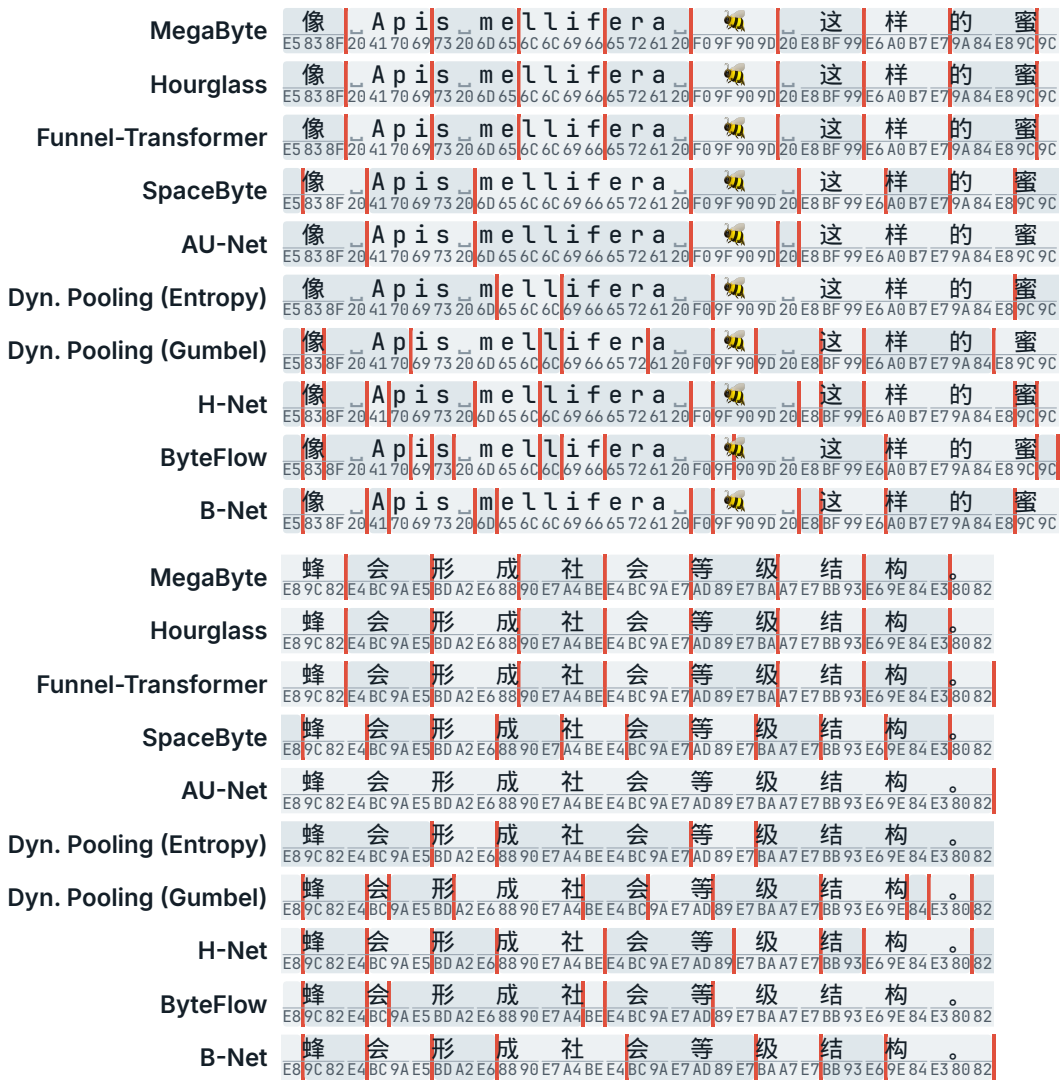


Figure 5: **Patch boundaries of the small Chinese hierarchical models on one sentence.** Rows follow Table 13; the BPE and byte Transformers are omitted. The layout follows Figure 4; each Chinese character takes three bytes and the emoji takes four.

10 of its 19 boundaries in the first 21 bytes. Its 9 boundaries in the Chinese text cut characters at varying offsets, 3 after a character's first byte, 4 after its second and 2 at its end. Its final patch spans the last 13 bytes, so the short early patches and long final span from Appendix G appear in Chinese as well.

Both figures come from running each final checkpoint on the displayed text.

D AUTOREGRESSIVE CAUSALITY

The downsampler, pooling, and upsampler must all preserve causality for a hierarchical Transformer to be trained with the same autoregressive next-byte objective as a standard causal Transformer. The logits at position t predict x_{t+1} and may depend only on the input prefix $x_{\leq t}$, even when training processes the full sequence in parallel with teacher forcing. The downsampler must therefore determine any boundary information used at position t from that prefix alone, without inspecting future tokens or representations. Pooling must ensure that a patch representation becomes available only after every input and boundary decision contributing to it is available. For example,

a mean or last-token representation of a completed patch cannot be used to predict tokens earlier within that patch. The upsampler must respect this availability when copying, projecting, or attending to patch representations, using appropriate causal shifts or masks to prevent information from being propagated backward to earlier predictions. These constraints apply at every hierarchy level, including the intervening sequence-processing modules, exponential moving average (EMA) smoothing, straight-through estimator (STE) gates, and residual connections. Satisfying these constraints ensures that the information available to each prediction during parallel training matches that available during autoregressive generation. Appendix A distinguishes causal convolutional scoring and threshold decisions from the sequence-wide ranking used during warmup and by the training patch-count safeguard.

Causality violation in AU-Net's downsampling rule. AU-Net (Videau et al., 2025, Section 2.2) requires its splitting function to preserve earlier pooling decisions when bytes are appended to the sequence. However, its released word-grouping rule violates this requirement. The regular expression `\s*[\r\n]` matches whitespace ending in a line break, and the implementation marks the start of that match as a pooling position.[1] Whether an earlier whitespace position becomes a boundary can therefore depend on a later newline.

For example, `one  x` and `one  \n` share their first five bytes, where `\n` denotes a single newline byte. At the two-word grouping level, the newline creates a boundary at the first space, the fourth byte, whereas the continuation with `x` does not. The boundary mask at that earlier position thus depends on the sixth byte. During parallel training, this mask determines which coarse representation is supplied to the finer stream, so predictions within the shared prefix can depend on the unseen continuation. Causal attention masks do not remove this dependence through boundary selection, and longer whitespace runs make any fixed byte delay insufficient. Consequently, AU-Net with this downsampling rule violates prefix causality; a causal replacement must determine boundaries from the observed prefix and delay their use until the deciding byte is available.

E Additional Evaluation Results

Table 4 reports the full small English evaluation, with a mean over all seven downstream tasks. B-Net achieves 44.6% mean accuracy, compared with 39.3% for MegaByte and 39.2% for Hourglass.

E.1 Textual Perturbations

We evaluate the large English models zero-shot on five perturbed HellaSwag variants, using the conditions reported by H-Net (Hwang et al., 2026): AntSpeak, Drop, RandomCase, Repeat, and UpperCase. Table 5 reports length-normalized accuracy for each condition and their unweighted mean; Appendix H.3 describes the evaluation and aggregation. B-Net achieves the highest reported average at 34.0%, compared with 33.3% for SpaceByte, 32.2% for H-Net, and 29.6% for the BPE reference. B-Net leads on Drop and UpperCase, SpaceByte on RandomCase and Repeat, and AU-Net on AntSpeak.

E.2 Qualitative Generation Examples

Table 6 shows one continuation per large English model under a shared prompt. These samples illustrate model outputs without establishing typical generation quality.

E.3 Validation Histories

Figures 6 and 7 show the small-English and Chinese validation histories.

F Component Ablations

Tables 7 and 8 report the completed small English substitutions and their task-level results. The reference uses convolutional boundary scoring with a window of four, adaptive threshold control,

[1] See `NWORD_PUNCT_RE` and `RegexPool.str_offset` in the released AU-Net implementation.

Table 4: **Zero-shot evaluation of small models across multiple benchmarks.** Models use the same training-data budget and approximately matched forward-compute estimates; see Section 4.1. Accuracy is reported in percent. Mean averages LAMBADA, HellaSwag, PIQA, ARC-Easy, ARC-Challenge, XWinograd-en, and WinoGrande. Wiki denotes WikiText; Dyn. Pool. denotes Dynamic Pooling. LAMBADA uses the OpenAI variant; XW-en is the English XWinograd subset. Random denotes uniform choice selection, with dashes for WikiText, LAMBADA, and the seven-task Mean. Best results are bold and second-best results are underlined, ranked before rounding.

Model	Wiki bpb ↓	LMB. acc ↑	Hella. acc_n ↑	PIQA acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	XW-en acc ↑	Wino. acc ↑	Mean acc ↑
Random	–	–	25.0	50.0	25.0	25.0	50.0	50.0	–
BPE Transformer	<u>1.01</u>	25.3	32.3	**62.8**	**52.6**	**25.9**	<u>60.9</u>	<u>51.5</u>	<u>44.4</u>
Byte Transformer	1.18	17.9	29.5	57.8	40.0	21.6	55.5	49.6	38.8
MegaByte	1.12	19.8	30.6	58.1	40.2	21.8	54.9	49.5	39.3
Hourglass	1.12	20.8	30.9	57.9	39.4	20.9	54.7	50.0	39.2
Funnel-Transformer	1.09	22.6	31.4	60.0	42.6	21.8	56.1	50.2	40.7
SpaceByte	1.05	**27.3**	32.0	60.3	44.3	23.2	60.6	**51.7**	42.8
AU-Net	1.05	26.3	32.0	59.8	45.3	24.4	58.5	50.9	42.4
Dyn. Pool. (Entropy)	1.10	22.2	31.5	59.0	42.6	24.0	56.6	51.3	41.0
Dyn. Pooling (Gumbel)	1.09	23.5	32.1	58.1	43.1	23.3	56.0	49.8	40.8
H-Net	1.01	21.4	<u>33.8</u>	60.3	47.3	24.8	59.4	50.6	42.5
ByteFlow	1.19	12.3	29.4	56.8	36.5	21.8	53.8	51.0	37.4
B-Net	**0.99**	<u>26.8</u>	**34.3**	<u>62.6</u>	<u>51.6</u>	<u>25.0</u>	**61.7**	50.5	**44.6**

Table 5: **Zero-shot HellaSwag accuracy under textual perturbations.** Five perturbation accuracies and their average use length-normalized accuracy, reported in percent. Random denotes uniform choice selection among the four endings, with a dash for Input. Best and second-best available results are bold and underlined, ranked before rounding.

Model	Input	HellaSwag					Average ↑
		AntSpeak	Drop	Random Case	Repeat	Upper Case	
Random	–	25.0	25.0	25.0	25.0	25.0	25.0
BPE Transformer	Token	30.9	28.0	26.1	26.8	36.1	29.6
Byte Transformer	Byte	<u>31.0</u>	26.4	27.4	27.3	35.0	29.4
MegaByte	Byte	30.7	28.3	30.9	27.5	41.4	31.8
Hourglass	Byte	30.9	27.9	31.0	27.7	41.5	31.8
Funnel-Transformer	Byte	30.9	28.4	30.3	27.9	41.9	31.9
SpaceByte	Byte	31.0	<u>28.8</u>	**32.7**	**28.3**	<u>45.9</u>	<u>33.3</u>
AU-Net	Byte	**31.2**	27.9	30.5	27.6	42.5	31.9
Dyn. Pooling (Entropy)	Byte	30.1	27.4	29.8	27.9	40.2	31.1
Dyn. Pooling (Gumbel)	Byte	30.6	28.1	29.7	27.7	41.5	31.5
H-Net	Byte	29.6	28.3	30.6	27.7	44.9	32.2
ByteFlow	Byte	28.0	26.2	27.1	25.8	28.2	27.0
B-Net	Byte	30.8	**29.5**	<u>31.1</u>	<u>28.2</u>	**50.2**	**34.0**

and EMA reconstruction without an auxiliary ratio loss or upsampling straight-through estimator. The space-like variant also removes EMA smoothing, so it changes more than the boundary rule. These comparisons support the component choices at small scale but do not establish local optimality over all possible module substitutions or their effects at larger scales.

Table 6: **Large-checkpoint generation examples.** Each row shows sample 1 after the shared prompt "Once upon a time". The first 14 continuation words are shown.

MODEL	CONTINUATION
BPE Transformer	there was a young man named John who was the first ever to have …
Byte Transformer	the post office in Asheville, North Carolina was called the "Blue Jacket" Post Office. …
MegaByte	there was a family whose name was Frederick. Frederick, the great grandfather of Seinfeld, …
Hourglass	there was a family whose name would be remembered throughout history. The family held …
Funnel-Transformer	there was a family whose name would be forever marked in the history of …
SpaceByte	there was a fair little bird called Steve. He was a friendly little guy …
AU-Net	a school board decided to take the social studies curriculum in grades two through …
Dyn. Pooling (Entropy)	there was a village just a few hundred miles from the eponymous state. In …
Dyn. Pooling (Gumbel)	most people would start off by behaving well. Even the very youngest children would …
H-Net	there was a farming state that occupied the entire present day French Alps, The …
ByteFlow	when yos his flight was building the old bean first through the rise and …
B-Net	there was a fairy. Her name was Furious. She was big, she was strong, …

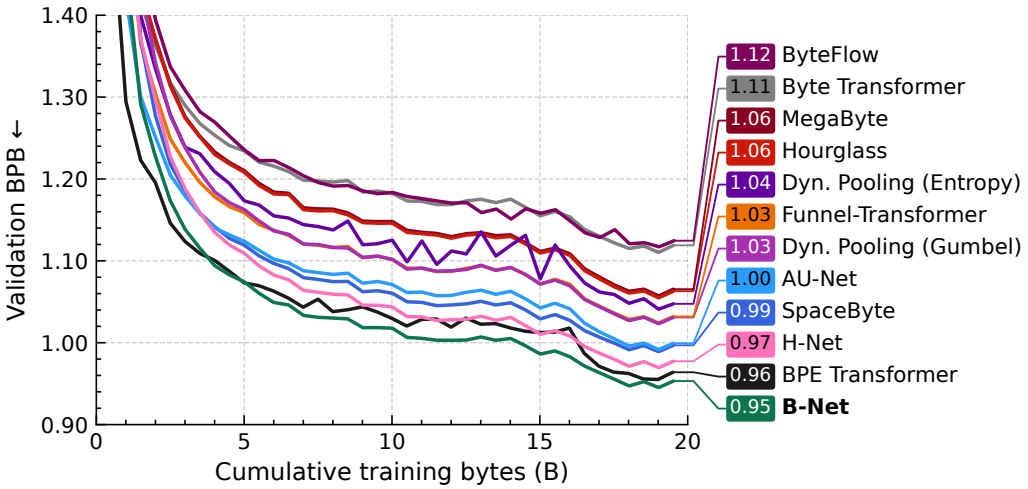


Figure 6: **Validation BPB of small models on FineWeb-Edu.** Curves show unsmoothed intermediate validation measurements against cumulative training bytes; lower is better. Line hues group models by patching mechanism and match the row shading in Table 4. Boxed values give each model's final BPB over the full validation set, and labels are ranked by the unrounded values.

G BYTEFLOW IMPLEMENTATION AND DIAGNOSTICS

Implementation scope. We implement ByteFlow (Deng et al., 2026) from its published description and report results from this reproduction. The implementation was checked against the paper, but agreement with the authors' implementation cannot be established without access to their code. We therefore distinguish observations in our runs from claims about the original implementation. At the large English scale, our reproduction reaches 1.05 WikiText bpb and 32.2% mean downstream accuracy (Table 1).

Boundary placement and patch count. ByteFlow selects a prescribed number of positions through Top-K, which constrains the compressed sequence length. This count does not require the selected positions to be distributed evenly across the input. In our reproduction, we observe short early patches followed by a long final span; Figure 8 reports the average patch length exclud-

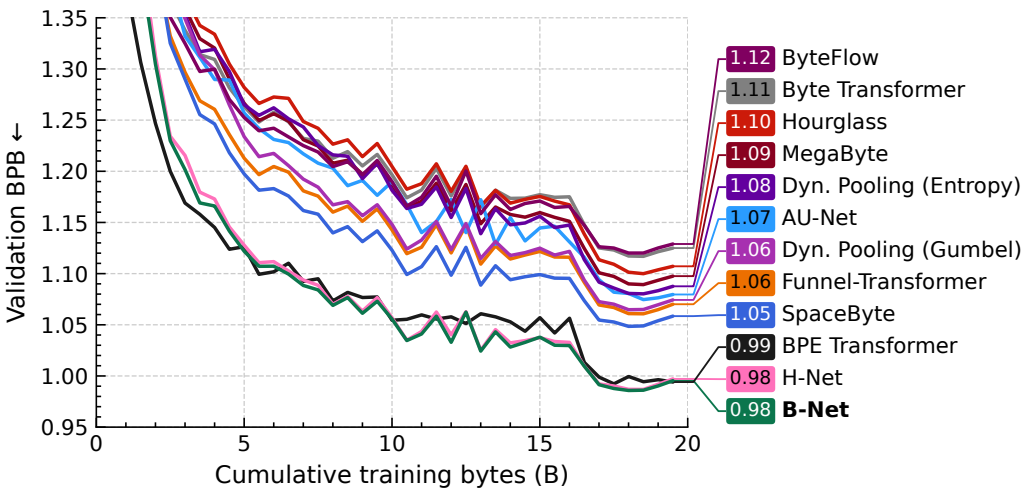


Figure 7: **Intermediate validation bpb on FineWeb-Edu-Chinese-V2.1.** The eleven completed runs have unsmoothed measurements at roughly 0.5-billion-byte intervals. Line hues group models by patching mechanism and match the row shading in Table 2.

Table 7: **Boundary-scorer and selection-rule ablation configurations.** Bold entries mark changes relative to B-Net. All variants use 8192-byte sequences and an M5 + T14 + M5 backbone with hidden dimensions 576/768, where T and M denote Transformer and Mamba-2 layers. The Space-Like variant additionally removes EMA smoothing. STE denotes the straight-through estimator used in upsampling.

Variant	Boundary scorer	Boundary rule	Ratio loss	Upsampling STE
B-Net	Convolution, Window 4	Adaptive Threshold	No	No
Convolution window 1	**Convolution, Window 1**	Adaptive Threshold	No	No
Ratio loss	Convolution, Window 4	**Fixed Threshold (0.5)**	**Yes**	No
Ratio loss + STE	Convolution, Window 4	**Fixed Threshold (0.5)**	**Yes**	**Yes**
Cosine scorer	**Cosine Similarity**	Adaptive Threshold	No	No
Space-like rule	**None**	**Space-Like Rule**	No	No

ing that final span. This diagnostic distinguishes uneven segmentation from a failure to maintain the selected count.

The excluded final span must be considered when interpreting this statistic: a fixed number of short early patches can coexist with a long tail while preserving the count-based compression ratio. The logged readings sample the first microbatch of each logging interval on rank zero, rather than aggregating every sequence processed during that interval. The current export does not quantify that tail's length or its distribution across examples, so it cannot establish the magnitude of this imbalance or its causal effect on predictive quality. These observations concern our paper-based reproduction and should not be attributed to unavailable official checkpoints. Concentrating boundaries early in the sequence leaves a long suffix without additional selected positions and may partly explain weaker predictive performance in our reproduction. Establishing this contribution would require a controlled comparison that changes boundary placement while holding the remaining setup fixed.

Training and inference causality. Causal attention masks prevent direct attention to future representations, but full-sequence Top-K selection can still make an earlier boundary decision depend on later scores. The ByteFlow authors clarify in their review responses that training uses full-sequence

Table 8: **Small-scale component ablations.** Models train on 20B bytes. Val. bpb is the validation bits per byte; patch length is the logged validation-batch mean; training FLOPs are setup estimates scaled by training bytes. Wiki denotes WikiText; XW-en denotes XWinograd-en. Accuracy is reported in percent. Mean averages LAMBADA, HellaSwag, PIQA, ARC-Easy, ARC-Challenge, XWinograd-en, and WinoGrande. All variants follow the same FLOP- and data-matching protocol, with estimated compute between 0.99 and 1.02 times the reference.

Module	Replacement	Val. bpb ↓	Patch len.	FLOPs / ref.	Wiki bpb ↓	LMB. acc ↑	Hella. acc_n ↑	PIQA acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	XW-en acc ↑	Wino. acc ↑	Mean acc ↑
Reference	B-Net	0.95	6.0	1.00	0.99	26.8	34.3	62.6	51.6	25.0	61.7	50.5	44.6
Boundary scorer	Pointwise scorer (w=1)	0.97	6.0	1.00	1.01	25.5	33.5	60.0	47.0	24.2	59.5	51.1	43.0
	Cosine similarity	0.97	6.1	1.01	1.01	23.6	34.0	61.8	48.6	24.4	59.4	49.3	43.0
Selection rule	Freeze threshold after warmup	0.95	5.8	1.00	0.99	27.9	34.2	61.3	49.0	25.3	60.2	50.0	44.0
	Ratio loss	0.97	5.9	1.00	1.02	22.6	33.7	62.2	47.0	24.7	60.1	48.9	42.7
	Ratio loss + upsampling STE	0.97	5.9	1.00	1.02	20.0	33.7	61.0	45.7	23.6	59.3	51.2	42.1
Downsampler	Fixed-stride boundaries	1.14	6.0	1.00	1.21	11.6	29.6	57.7	38.6	21.2	53.6	49.8	37.4
	Space-like boundaries	0.95	6.0	1.00	0.99	29.1	34.5	62.3	49.7	24.1	61.4	50.6	44.5
Pooling	Mean	0.97	6.1	1.00	1.01	24.4	34.1	61.2	48.2	23.6	60.5	52.3	43.5
	Sum	0.98	6.0	1.00	1.02	24.2	33.0	61.9	46.3	25.1	59.4	48.3	42.6
	Within-patch attention	0.97	6.1	1.02	1.02	26.6	33.5	61.8	46.7	23.1	61.3	50.7	43.4
Upsampler	Learned projection	1.15	6.0	1.01	1.21	10.9	28.8	58.0	37.8	22.1	55.0	48.5	37.3
Downsampling mixer	Causal attention	0.97	6.0	0.99	1.02	23.2	33.6	61.0	49.6	23.5	60.0	48.7	42.8
Upsampling mixer	Causal attention	0.97	6.0	1.01	1.01	24.5	33.6	61.5	48.1	24.1	61.1	50.0	43.3
Smoother	Identity + upsampling STE	1.14	4.4	1.00	1.20	11.4	29.4	57.5	37.6	21.0	55.3	49.5	37.4
Encoder/decoder mixer	Transformer blocks only (3 layers)	1.18	6.1	1.01	1.25	15.8	29.2	57.0	37.0	20.4	52.2	51.5	37.6
	Transformer blocks only (4 layers)	1.12	5.9	1.00	1.19	18.2	29.6	58.4	41.0	21.9	54.4	48.5	38.9

selection, whereas evaluation uses rolling Top-K over already observed positions.[2] This distinction permits causal inference without establishing prefix-causal training; it does not justify attributing noncausal evaluation to their reported results.

Scope of published comparisons. ByteFlow (Deng et al., 2026, Section 4.4) compares alternative chunkers within its own architecture, so the H-Net- and BLT-inspired variants do not constitute comparisons against those complete models. The paper also reports a separately trained SpaceByte

[2]ByteFlow author responses on training-time selection and autoregressive evaluation.

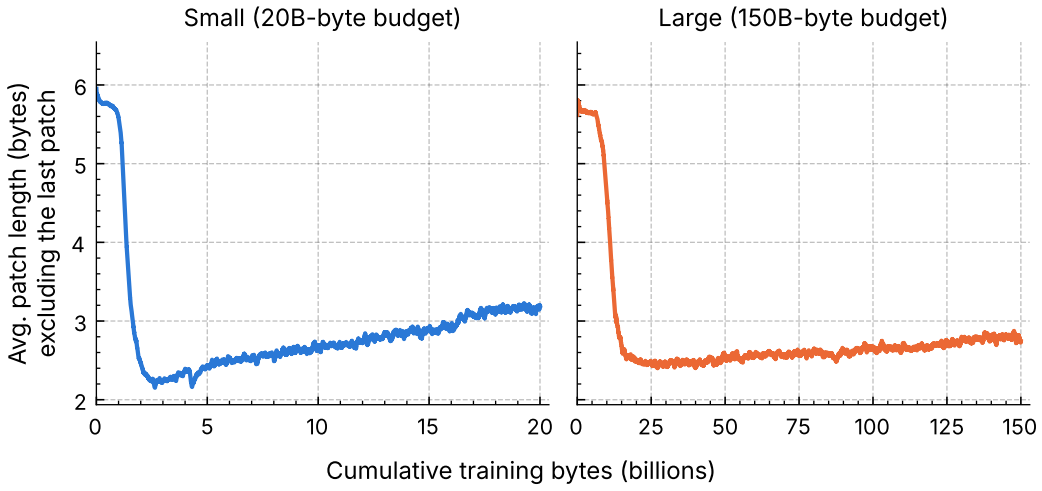


Figure 8: **Patch-length diagnostic for our ByteFlow reproduction.** The left panel shows the small model (blue) and the right panel the large model (orange), against cumulative training bytes. The logged mean excludes the final trailing span; its decline indicates shorter non-final patches, rather than a change in the fixed Top-K count or a compute-budget overrun. Curves use trailing means with windows corresponding to approximately 0.5% of each training budget. Panel titles specify the respective training budgets.

baseline, although its word-boundary description in Section 4.2 conflicts with the entropy-based description in Appendix C.3.2. These details limit what can be inferred about the original-model comparisons; they do not explain the performance of our reproduction. Our comparison therefore rests on the implementations and compute budgets evaluated here, with the segmentation diagnostic providing a possible explanation rather than a causal attribution.

H Architecture and Model Configurations

H.1 General Framework and Architecture Operations

Modular architecture. A downsampler partitions fine-resolution representations into contiguous patches, pooling constructs one representation per patch, and a coarse processing stack operates on the shorter sequence. An upsampler combines causally available coarse representations with the fine-resolution stream to produce next-byte predictions. Figure 9 illustrates this general framework; each component admits different implementations. Instantiating these components yields the architectures summarized in Table 9 and the model configurations below.

B-Net instantiates the framework with Mamba-2 outer stacks and a Transformer inner stack, retaining H-Net's hierarchical processing design while replacing its patching mechanism with convolutional boundary scoring and adaptive threshold control. It uses EMA reconstruction with copy upsampling, as described in Section 3.

Figure 10 illustrates the operations used to construct hierarchical models. These operations admit different implementations, so Table 9 identifies the choices associated with each architecture.

H.2 Architecture Component Mapping

Table 9 expresses representative configurations of prior architectures in terms of the framework's components. Several works also evaluate alternative pooling and upsampling choices. Our benchmark tables use Hourglass with concatenation pooling and deconcatenation, whereas the mapping below illustrates its attention-based variant.

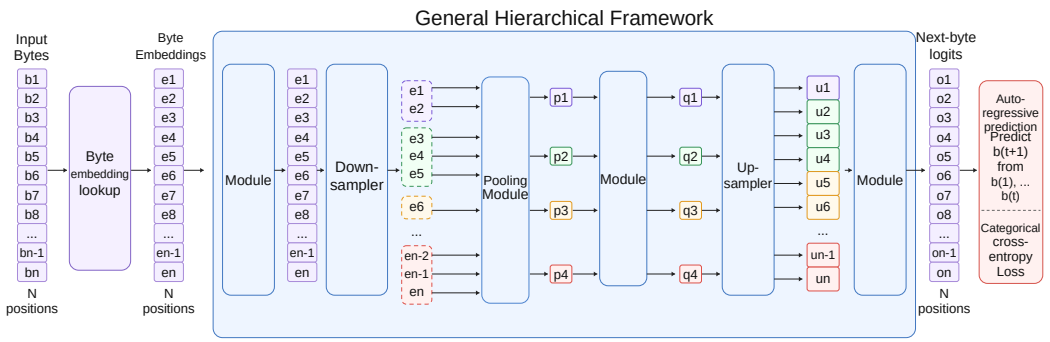


Figure 9: General framework for hierarchical byte-level models. Processing modules operate before compression, on the coarse sequence, and after reconstruction. Downsampling selects patch boundaries, pooling forms patch representations, and upsampling restores the finer resolution. The blocks represent interchangeable components, rather than a specific model; Figure 10 illustrates several choices. The patch groups are schematic, and autoregressive implementations must preserve causal representation availability.

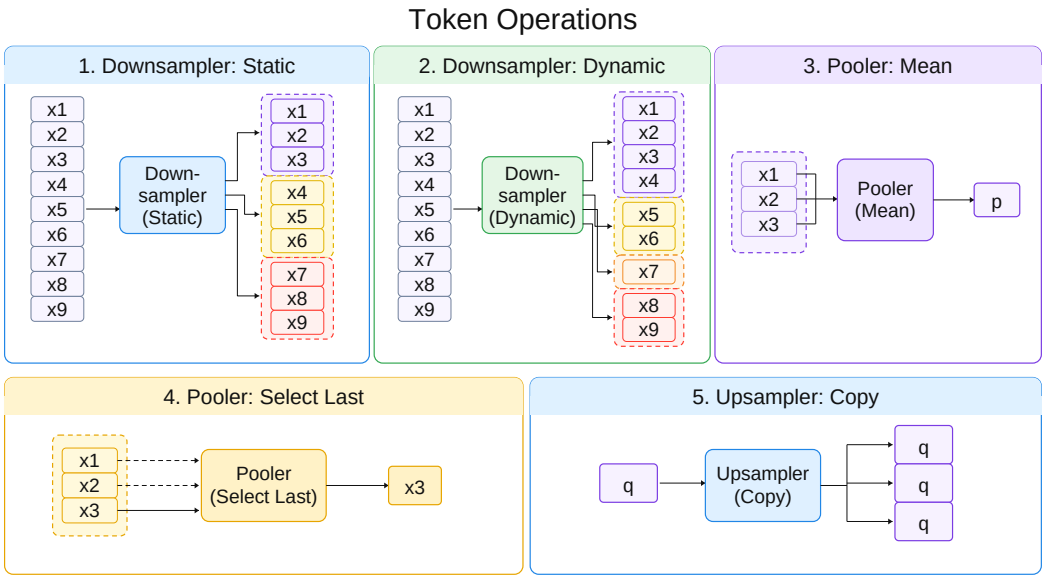


Figure 10: Downsampling, pooling, and upsampling operations. Static and dynamic downsamplers partition the sequence; mean and last-position pooling construct patch representations; copy upsampling restores the finer resolution. Autoregressive use requires the causal alignment described in Appendix D.

For MegaByte, the upsampling entry denotes the closest framework analogue: its global patch representation is reshaped and projected into position-specific inputs for the local autoregressive model. Funnel-Transformer is a bidirectional encoder architecture, but we include it because its compression and reconstruction mechanisms are direct precursors to causal hierarchies such as Hourglass.

H.3 Model Configurations and Evaluation Details

BLT comparison scope. We do not include a direct BLT comparison because we could not establish a reliable reproduction using the released implementation and documented settings. Related reproduction difficulties have been reported in the public issue tracker, and the repository notes

Table 9: Representative architectures expressed using the components of our general framework.

Model	Downsampler	Pooling	Upsampler	Other Modules
MegaByte (Yu et al., 2023)	Fixed	Concatenate patch embeddings	Reshape into slots; shared projection	Transformer blocks
Hourglass (Nawrot et al., 2022)	Fixed	Attention with mean seed	Attention with linear seed	Transformer blocks
Funnel-Transformer (Dai et al., 2020)	Fixed	Strided mean	One-shot repeat	Transformer blocks
SpaceByte (Slagle, 2024)	Space-like byte-boundary rules	Select boundary state	Patch-end addition	Transformer blocks
AU-Net (Videau et al., 2025)	Regex boundaries; deeper word-count grouping with sentence-end splits	Select and linearly project	Multi-linear copy	Transformer blocks
Dyn. Pooling (Entropy) (Nawrot et al., 2023)	Entropy-based boundaries	Average over segment	Causal copy	Transformer blocks
Dyn. Pooling (Gumbel) (Nawrot et al., 2023)	Gumbel-based boundaries	Average over segment	Causal copy	Transformer blocks
H-Net (Hwang et al., 2026)	Cosine-similarity boundaries	Select boundary states	Confidence-gated copy (STE)	Transformer blocks; Mamba blocks
ByteFlow (Deng et al., 2026)	Coding-rate Top-K boundaries	Select and linearly project	Position-specific linear copy with residual	Transformer blocks; sliding-window attention; Canon layers

ongoing work to facilitate reproduction of the published results.[3] H-Net (Hwang et al., 2026, Appendix A.1.3) likewise omits BLT, citing the implementation complexity of its auxiliary entropy model, calibrated patching heuristics, and hash embeddings. We therefore omit BLT to avoid drawing conclusions from an unverified reproduction. As supporting context, AU-Net (Videau et al., 2025, Table 2) reports higher scores than BLT on all four jointly reported benchmarks at a similar reported compute budget. H-Net, SpaceByte, and AU-Net provide strong comparisons spanning learned and rule-based patching, evaluated alongside the other baselines under our FLOP- and data-matching protocol.

Configurations. Tables 11, 12, and 13 give the current large English, small English, and Chinese configurations. The FLOP values are static forward-pass estimates under the protocol in Section 4.1. Compression values summarize representative average bytes per inner position (bytes per token for BPE), rounded to one decimal; learned-patcher values come from held-out evaluation or late-training averages. Table 10 summarizes the training settings and B-Net controller parameters. The warmup–stable–decay (WSD) learning-rate schedule uses linear warmup, a constant-rate phase, and an inverse-square-root decay phase.

[3] See the BLT repository's development-status statement and reports concerning validation-loss stagnation, patch-length discrepancies, and entropy-context reset details.

Evaluation harness. We use version 0.4.11 of the Language Model Evaluation Harness (Gao et al., 2024). All reported tasks use zero shots. The packaged task revisions are 1.0 for ARC-Easy, ARC-Challenge, HellaSwag, PIQA, LAMBADA-OpenAI, WinoGrande, and XWinograd-en/zh, and 2.0 for WikiText.

HellaSwag perturbation protocol. Following H-Net (Hwang et al., 2026), we evaluate five HellaSwag text perturbations at zero shots. Our revision-1.0 task definitions apply each transformation to the preprocessed query and all four answer choices on the validation split. AntSpeak uppercases text and inserts spaces between adjacent characters within each non-whitespace span, preserving existing whitespace. Drop independently deletes each character with probability 0.1, including whitespace. RandomCase independently uppercases or lowercases each character with equal probability. Repeat independently selects each character with probability 0.2 and adds one to four copies, sampled uniformly, including whitespace. UpperCase converts all text to uppercase. We report length-normalized accuracy (`acc_norm`) in percent and compute the unweighted mean over all five conditions before rounding.

Table 10: **Training and B-Net controller settings.** Batch sizes count sequences across devices and gradient-accumulation steps. Controller settings apply to B-Net; its target-rate sweep varies the requested patch length.

Setting	Large English	Small English / Chinese
Training bytes	150B	20B
Byte-model context	8192 bytes	8192 bytes
Global batch size	512	64
Optimizer	AdamW	AdamW
Peak learning rate	6.25×10^{-4}	1.5×10^{-3}
Adam coefficients	$(0.9, 0.95)$	$(0.9, 0.95)$
Weight decay	0.01	0.01
Gradient-norm clipping	1.0	1.0
Mixed precision	BF16	BF16
Learning-rate schedule	WSD	WSD
Learning-rate warmup	10% of training	10% of training
Learning-rate decay phase	Final 20%	Final 20%
B-Net scoring window	4 bytes	4 bytes
Threshold EMA decay	0.999	0.999
Top-K warmup (optimizer steps)	3578	3816
Capacity tolerance	25%	25%
Capacity mode	Maximum-only	Maximum-only

Table 11: **Large model architectures.** Models use the BPE reference training-data budget and approximately match its estimated forward FLOPs per byte. Hierarchical models use a single downsampling stage, with the downsampler listed in the Tokenizer column. The BPE reference uses 1792 tokens; all other models use 8192 bytes. Comp. denotes representative average compression. T and M denote Transformer and Mamba-2 blocks; C denotes a Transformer block augmented with Canon layers.

Model	Tokenizer	Comp.	Layers	Hidden dim.	Params	GFLOPs / byte
BPE Transformer	GPT2	4.6	T24	1536	757M	0.42
Byte Transformer	—	1.0	T15	1024	193M	0.45
MegaByte	Fixed	8.0	T8 + T16 + T8	768, 1536	581M	0.41
Hourglass	Fixed	8.0	T8 + T17 + T8	768, 1536	609M	0.42
Funnel-Transformer	Fixed	6.0	T8 + T13 + T8	768, 1536	484M	0.42
SpaceByte	Space-Like	6.0	T8 + T16 + T8	768, 1536	586M	0.45
AU-Net	Regex	5.2	T8 + T12 + T8	768, 1536	468M	0.44
Dyn. Pooling (Entropy)	Entropy	6.4	T8 + T14 + T8	768, 1536	510M	0.42
Dyn. Pooling (Gumbel)	Gumbel	6.0	T8 + T14 + T8	768, 1536	510M	0.43
H-Net	Cosine Similarity + Ratio Loss	6.0	M5 + T25 + M5	1024, 1536	778M	0.43
ByteFlow	Coding-Rate + Top-K	6.0	C6 + T29 + C6	512, 1536	876M	0.43
B-Net	Convolution + Adaptive Threshold	6.0	M5 + T25 + M5	1024, 1536	775M	0.42

Table 12: **Small model architectures.** Models use the BPE reference training-data budget and approximately match its estimated forward FLOPs per byte. Hierarchical models use a single downsampling stage, with the downsamplers listed in the Tokenizer column. The BPE reference uses 1792 tokens; all other models use 8192 bytes. Comp. denotes representative average compression. T and M denote Transformer and Mamba-2 blocks; C denotes a Transformer block augmented with Canon layers.

Model	Tokenizer	Comp.	Layers	Hidden dim.	Params	GFLOPs / byte
BPE Transformer	GPT2	4.6	T12	768	124M	0.09
Byte Transformer	—	1.0	T8	576	32M	0.08
MegaByte	Fixed	8.0	T5 + T15 + T5	384, 768	128M	0.09
Hourglass	Fixed	8.0	T5 + T15 + T5	384, 768	128M	0.09
Funnel-Transformer	Fixed	6.0	T5 + T11 + T5	384, 768	96M	0.09
SpaceByte	Space-Like	6.0	T5 + T11 + T5	384, 768	105M	0.09
AU-Net	Regex	5.2	T5 + T9 + T5	384, 768	85M	0.09
Dyn. Pooling (Entropy)	Entropy	5.5	T5 + T11 + T5	384, 768	96M	0.09
Dyn. Pooling (Gumbel)	Gumbel	6.0	T5 + T11 + T5	384, 768	96M	0.09
H-Net	Cosine Similarity + Ratio Loss	6.0	M5 + T14 + M5	576, 768	121M	0.09
ByteFlow	Coding-Rate + Top-K	6.0	C5 + T11 + C5	384, 768	101M	0.09
B-Net	Convolution + Adaptive Threshold	6.0	M5 + T14 + M5	576, 768	121M	0.09

Table 13: **Chinese model architectures.** Models train on 20B bytes of FineWeb-Edu-Chinese-V2.1. The listed FLOPs per byte are forward-compute estimates at the reported compression rates, not accumulated training measurements. The BPE reference uses 1792 tokens; all other models use 8192 bytes. Comp. denotes representative average compression. T and M denote Transformer and Mamba-2 blocks; C denotes a Transformer block augmented with Canon layers.

Model	Tokenizer	Comp.	Layers	Hidden dim.	Params	GFLOPs / byte
BPE Transformer	Qwen2.5	4.6	T12	512	116M	0.10
Byte Transformer	—	1.0	T9	576	36M	0.09
MegaByte	Fixed	4.0	T5 + T6 + T5	384, 768	62M	0.08
Hourglass	Fixed	4.0	T5 + T5 + T5	384, 768	56M	0.08
Funnel-Transformer	Fixed	4.0	T5 + T6 + T5	384, 768	61M	0.08
SpaceByte	Space-Like	3.5	T5 + T5 + T5	384, 768	63M	0.09
AU-Net	Regex	16.7	T5 + T35 + T5	384, 768	276M	0.09
Dyn. Pooling (Entropy)	Entropy	5.7	T5 + T11 + T5	384, 768	96M	0.09
Dyn. Pooling (Gumbel)	Gumbel	3.5	T5 + T5 + T5	384, 768	53M	0.08
H-Net	Cosine Similarity + Ratio Loss	3.5	M4 + T8 + M4	576, 768	75M	0.09
ByteFlow	Coding-Rate + Top-K	3.5	C5 + T6 + C5	384, 768	65M	0.09
B-Net	Convolution + Adaptive Threshold	3.5	M4 + T8 + M4	576, 768	74M	0.09

I FLOP Counting Methodology

This appendix specifies the operation counts used for the forward-pass estimates in Table 11 and the compute-matching protocol in Section 4. We first define the counting conventions and shared primitives, then give the costs of sequence processing, boundary prediction, pooling, and reconstruction. These are analytical estimates under the implementation's counting conventions; Section I.7 describes their limitations.

I.1 Counting Conventions and Notation

Units and scope. All formulas count operations for one input sequence (batch size one). A multiply and an add count as two FLOPs, so multiplying matrices of shapes (m, k) and (k, n) costs $2mkn$. The per-byte values in Table 11 divide the sequence-level forward count by the assumed number of input bytes and by 10^9. Backward costs are assigned separately: twice the forward cost for matrix-multiplication terms, the same cost for elementwise terms, and zero for certain parameter-free reductions. The resulting forward-plus-backward estimate is therefore approximately, but not exactly, three times the forward count.

Sequence lengths. We write L for the expected sequence length at a module and d for its feature width, corresponding to S and D in Section 3 at a single hierarchy level. After downsampling, the expected length is $N \approx L/p$, where p is the expected patch length. For variable-length sequences, the counter rounds expected dimensions up at the point of use; ceilings are shown explicitly where relevant. Table 14 summarizes the notation.

Linear projection. We express projection costs using a shared primitive,

$$\mathrm{Lin}(d_{\mathrm{in}}, d_{\mathrm{out}}, L, b) = 2Ld_{\mathrm{in}}d_{\mathrm{out}} + bLd_{\mathrm{out}}, \tag{8}$$

where the two terms account for the matrix multiplication and optional bias addition. Named module terms below denote forward FLOP costs and add directly when modules are composed.

Table 14: Notation for per-sequence forward FLOP counts.

Symbol	Meaning
L, N	Expected sequence lengths before and after downsampling
p	Expected tokens per patch, with $N \approx L/p$
d, d_{in}, d_{out}	Feature width, or input and output widths when they differ
V	Vocabulary size: 258 for bytes and 50,257 for English GPT-2 BPE
h	Feed-forward hidden width
H	Number of attention heads
d_{qk}, d_v	Total query/key and value widths across all heads
L_q, L_k, d_{ctx}	Query length, key/value length, and context feature width
A	Number of query–key pairs charged by the attention counter
w	Attention window size or convolution kernel size
b	Bias indicator: 1 if present, 0 otherwise

I.2 Token-Level Modules

Embedding and output head. The counter treats a learned embedding as a dense multiplication by a one-hot input:

$$C_{\text{embed}} = 2Vd\lceil L \rceil. \tag{9}$$

A one-hot embedding itself costs zero. The bias-free vocabulary output head has the same matrix-multiplication cost, $2LdV$. The dense-embedding convention has a larger effect for the BPE vocabulary than for the byte vocabulary; it does not represent the cost of an embedding lookup.

Normalization and activation. For a sequence of length L and width d, our counter assigns $4Ld$ to RMSNorm (Zhang & Sennrich, 2019) and $3Ld$ to LayerNorm (Ba et al., 2016) and GroupNorm (Wu & He, 2018). A monotone pointwise activation, such as ReLU or sigmoid, costs Ld; swish costs $2Ld$. Identity normalization and activation cost zero. We write $\text{Act}(L, h)$ for the activation cost at hidden width h.

Positional encoding. RoPE (Su et al., 2024) is assigned the per-call constant $d_{\text{rope head}} + \lfloor d_{\text{rope head}}/2 \rfloor$, where $d_{\text{rope head}}$ is the rotary dimension of one head. This count is independent of sequence length and head count, and makes positional encoding negligible in the estimate, consistent with the convention adopted from H-Net (Hwang et al., 2026). Learned and identity positional encodings are assigned zero cost.

GLU feed-forward network. For the GLU feed-forward network (Shazeer, 2020), the fused gate and up projection has width $2h$, followed by a down projection to d_{out}:

$$\begin{aligned} C_{\text{FFN}} = {} & \text{Lin}(d, 2h, L, b) + \text{Lin}(h, d_{\text{out}}, L, b) \\ & + 5Lh + \text{Act}(L, h). \end{aligned} \tag{10}$$

The counter assigns $5Lh$ to the remaining gating operations. With swish, the last two terms sum to $7Lh$.

Transformer block. A block combines normalization, a sequence mixer, an optional feed-forward network, residual additions, and any configured Canon layers:

$$\begin{aligned} C_{\text{block}} = {} & \text{PreNorm} + \text{Mixer} + \text{Canon}_{\text{mix}} \\ & + [\text{PostNorm} + \text{FFN} + \text{Canon}_{\text{ffn}}] + rLd. \end{aligned} \tag{11}$$

The bracketed terms are included when an FFN is present; $r = 2$ in that case and $r = 1$ otherwise. The mixer is attention (Section I.3) or Mamba-2. Canon terms are zero when the corresponding layer is absent.

Canon layer (ByteFlow). Each Canon layer is a depthwise causal convolution with kernel size k and forward cost

$$C_{\text{Canon}} = 2kd\lceil L \rceil. \tag{12}$$

The counter assigns twice this cost to its backward pass. Each Transformer block in ByteFlow's local encoder and decoder applies one Canon layer after the attention residual and one after the FFN residual; the reported configurations use $k = 4$.

Mamba-2 mixer. The Mamba-2 count separates projections, the state-space recurrence, convolution, and gating:

$$\begin{aligned} C_{\text{Mamba2}} = {} & 4Ld_{\text{model}}d_{\text{inner}} && X, Z \text{ projections} \\ & + 2Ld_{\text{model}}(2d_{\text{state}} + n_{\text{heads}}) && B, C, \Delta t \text{ projections} \\ & + 6Ld_{\text{inner}}d_{\text{state}} && \text{state-space recurrence} \\ & + 2Ld_{\text{inner}}d_{\text{conv}} && \text{depthwise convolution} \\ & + 5Ld_{\text{inner}} && \text{gating} \\ & + 2Ld_{\text{inner}}d_{\text{model}} && \text{output projection.} \end{aligned} \tag{13}$$

Here d_{model} is the input width, $d_{\text{inner}} = \text{expand} \cdot d_{\text{model}}$, d_{state} is the state dimension, d_{conv} is the convolution kernel size, and $n_{\text{heads}} = d_{\text{inner}}/\text{headdim}$. This mixer is used in the outer stages of the evaluated H-Net and B-Net configurations.

Feature changes and composition. Padding or truncating features costs zero forward FLOPs. If widening appends a learnable vector, it incurs a backward cost of $(d_{\text{out}} - d_{\text{in}})L$. Repeating a module n times multiplies its cost by n, and an identity module costs zero. Parallel composition adds the splitter, branch, and recombination costs: a fixed or copy splitter is free, while a learned splitter costs $\text{Lin}(d, d_1 + d_2, L, b)$ for branch widths d_1 and d_2. No configuration in Table 11 uses parallel composition.

I.3 Attention

The same routine counts self-attention and attention-based upsampling by accepting separate query and key/value lengths. For $A = L_qL_k - M$ charged query–key pairs, where M is the number excluded by the counter,

$$\begin{aligned} C_{\text{attn}} = {} & \text{Lin}(d_{\text{in}}, d_{qk}, L_q, b) \\ & + \text{Lin}(d_{\text{ctx}}, d_{qk}, L_k, b) + \text{Lin}(d_{\text{ctx}}, d_v, L_k, b) \\ & + 2Ad_{qk} + 3AH + 2Ad_v \\ & + \text{Lin}(d_v, d_{\text{out}}, L_q, b). \end{aligned} \tag{14}$$

The first two lines contain the query, key, and value projections. The third counts the $QK^\top$ product, softmax, and attention-weighted value product; the last is the output projection.

Full and windowed attention. For full self-attention, $L_q = L_k = \lceil L \rceil$ and $M = 0$, giving $A = \lceil L \rceil^2$. For windowed self-attention, the counter sets $M = \lceil L^2 - Lw \rceil$, giving $A \approx Lw$ up to rounding. Following the counting convention of H-Net, causal masking does not reduce the full-attention term to approximately $L^2/2$. Thus, the pair count represents the nominal full or windowed attention pattern, rather than the precise number of causally valid pairs. The byte-level attention stages in our configurations use sliding windows; the patch-level attention stages use full attention.

I.4 Hierarchical Composition

The cost of a hierarchical module follows its structure in Section 3:

$$\begin{aligned} C_{\text{hier}}(L) = {} & \textstyle\sum \text{PreModules}(L) + \text{Downsampler}(L) + \text{DownsampleMixer} \\ & + \textstyle\sum \text{ResidualModules}(L) \\ & + \textstyle\sum \text{SubModules}(N) \\ & + \text{Upsampler}(L) + \text{UpsampleMixer} \\ & + \textstyle\sum \text{PostModules}(L). \end{aligned} \tag{15}$$

The nested sub-stack is evaluated recursively at $N \approx L/p$ positions. Its linear-in-length terms therefore scale approximately as $1/p$, and its full-attention pair terms as $1/p^2$, relative to processing the same modules at length L. These savings must be weighed against the surrounding downsampling and reconstruction costs. Each hierarchical configuration in Table 11 applies this recursion once. Their downsample and upsample mixers are identities and contribute zero FLOPs.

I.5 Boundary Prediction and Selection

A downsampler combines boundary prediction, hard-boundary selection, pooling, and any patch-level loss:

$$\begin{aligned} C_{\text{down}} = \text{Scorer}(L) + \text{BoundaryRule}(L) \\ + \text{Pooling}(\lceil N \rceil, \lceil p \rceil) + \text{PatchLoss}(\lceil L \rceil). \end{aligned} \tag{16}$$

The scorer term includes its configured auxiliary losses. Fixed-stride and byte-value rules, such as space-like or character boundaries, are assigned zero prediction and selection cost. The learned scorers have the following costs.

Similarity scorer (H-Net). Query and key projections are followed by normalized dot products between adjacent positions:

$$C_{\text{sim}} = \text{Lin}_q + \text{Lin}_k + 6Ld_{\text{proj}} + \textstyle\sum_i \text{Loss}_i(L). \tag{17}$$

Here d_{proj} is the projection width; Lin_q and Lin_k are the corresponding linear-projection costs. Under the counting convention of H-Net, the $6Ld_{\text{proj}}$ term comprises $4Ld_{\text{proj}}$ for two ℓ_2 normalizations and $2Ld_{\text{proj}}$ for the dot products. This scorer is used by H-Net.

Convolutional scorer. The causal convolution of Section 3 maps d input features to one boundary logit with kernel size w:

$$C_{\text{Conv}} = 2dw\lceil L \rceil + b\lceil L \rceil + \textstyle\sum_i \text{Loss}_i(\lceil L \rceil). \tag{18}$$

There is no additional output-width factor, and the bias is disabled by default. This is the scorer used by B-Net and its convolutional-scorer ablations.

Gumbel scorer. The Gumbel variant of the Dynamic Pooling Transformer (Nawrot et al., 2023) predicts one logit per position:

$$C_{\text{Gumbel}} = \text{Lin}(d, 1, L, b) + \textstyle\sum_i \text{Loss}_i(L). \tag{19}$$

Its projection costs $2dL$ when the bias is disabled.

Entropy scorer. The entropy-supervised Dynamic Pooling variant adds a vocabulary-sized prediction head to its boundary head:

$$\begin{aligned} C_{\text{entropy}} &= \text{Lin}(d, 1, L, b) + \text{Lin}(d, V, L, b) + 4LV \\ &\approx 2dVL + 4LV. \end{aligned} \tag{20}$$

The three terms are the boundary projection, vocabulary projection, and entropy computation. Its internal decision rule is $\sigma(\text{logits}) \geq 0.5$. The auxiliary vocabulary head makes this scorer more expensive than the other boundary predictors; its upsampler-side cross-entropy is omitted from the counter.

Coding-rate scorer (ByteFlow). For the exact log-determinant scorer, let $T = \lceil L \rceil$, let B be the block size, and let $J = \lceil T/B \rceil$. The forward count is

$$C_{\text{CR}} = J\big(4Bd^2 + \lfloor d^3/3 \rfloor + 2B^2d + \lfloor B^3/3 \rfloor\big) + 3T. \tag{21}$$

The blockwise terms count Gram-matrix products, Cholesky factorizations, triangular solves, and block-local products; $3T$ counts the score-to-probability conversion. The reported ByteFlow configurations use $B = 256$. The scorer detaches its inputs, so its backward FLOP count is zero.

I.5.1 HARD-BOUNDARY RULES

The similarity, convolutional, and Gumbel scorers use a separately configured rule to turn scores into hard boundaries. Fixed thresholding is assigned zero cost. The counter also assigns zero selection FLOPs to ByteFlow's Top-K rule, omitting the cost of ranking candidates. For adaptive thresholding, the counter assigns

$$C_{\mathrm{AT}} = 6(\lceil L \rceil - 1) \tag{22}$$

forward FLOPs and zero backward FLOPs, assuming $L - 1$ adjacent candidate positions. This is a bookkeeping approximation for the cutoff and threshold update in Section 3.2 and the training patch-count safeguard in Appendix A; it does not model the full cost of ranking candidates.

Combining the convolutional scorer and adaptive threshold gives the stage cost

$$\begin{aligned} C_{\mathrm{Conv+AT}} = {} & 2dw\lceil L \rceil + b\lceil L \rceil + 6(\lceil L \rceil - 1) \\ & + \textstyle\sum_i \mathrm{Loss}_i(\lceil L \rceil) \\ & + \mathrm{Pooling}(\lceil N \rceil, \lceil p \rceil) + \mathrm{PatchLoss}(\lceil L \rceil). \end{aligned} \tag{23}$$

Under this counting approximation, the predictor and selection terms are linear in L for fixed d and w, with the convolution contributing $2dw\lceil L \rceil$. Their direct cost does not depend on the resulting patch length. The threshold rule affects the model's larger compute terms through N, the length of the nested stage in Equation 15.

I.5.2 AUXILIARY DOWNSAMPLER LOSSES

Each configured scorer loss adds a forward term at its input length L. Table 15 lists the losses used by the comparison configurations.

Table 15: Auxiliary scorer losses, included in $\sum_i \mathrm{Loss}_i(L)$.

Loss	Forward FLOPs	Configurations
Ratio	$L + 8$	H-Net and its convolutional-predictor variant
Binomial prior	$4L + 20$	Dynamic Pooling, Gumbel

The convolutional scorer with adaptive thresholding has no auxiliary scorer loss: the controller replaces the ratio loss used with fixed thresholding. The entropy-supervised variant likewise has no separately configured scorer loss; its decision uses the entropy head in Equation 20.

I.6 POOLING AND RECONSTRUCTION

I.6.1 POOLING FUNCTIONS

Pooling processes N patches of expected length p, with $Np \approx L$. The common operations and their costs are summarized in Table 16.

Table 16: Pooling costs before rounding N and p to integer dimensions.

Pooling	Forward FLOPs	Configurations
Select	0	H-Net variants, SpaceByte, AU-Net
Mean	$Npd \approx Ld$	Hourglass (mean), Funnel-Transformer, Dynamic Pooling variants
Concatenation	$\mathrm{Lin}(pd, d_{\mathrm{out}}, N, b)$	MegaByte, Hourglass (concatenation)

Select pooling gathers the patch-final token. Concatenation itself is free; its listed cost is the subsequent projection, which is zero when that projection is the identity. Convolutional pooling is implemented but unused in Table 11; for kernel size w, it costs $2Nwd_{\mathrm{in}}d_{\mathrm{out}} + bNd_{\mathrm{out}}$.

I.6.2 UPSAMPLERS

An upsampler restores length L, optionally smooths transitions, and combines the reconstructed stream with the residual. We denote these shared costs by $\mathrm{Smooth}(L, d)$ and $\mathrm{Reduce}(L, d)$. The plain sum and straight-through sum (`sum_ste`) reductions both cost Ld forward FLOPs and are assigned zero backward cost; replacement costs zero. The EMA smoother costs $3Ld$ in sequential form or $2L^2d$ in tensorized form, while identity smoothing costs zero.

Copy and multi-linear. Repeating each patch vector across its tokens is free, giving

$$C_{\text{copy}} = \mathrm{Smooth}(L, d) + \mathrm{Reduce}(L, d).$$

H-Net uses EMA smoothing and a straight-through sum; Funnel-Transformer and both Dynamic Pooling variants also use copy upsampling. AU-Net's multi-linear upsampler adds a projection, $\mathrm{Lin}(d_{\text{in}}, d_{\text{out}}, L, b)$, to the copy cost.

Deconcatenation. MegaByte and Hourglass with concatenation pooling project each patch vector to a flattened output of width $d_{\text{flat}} = p\, d_{\text{out}}$, then split it across token positions. For input patch width d_{ds}, the cost is

$$\begin{aligned} C_{\text{deconcat}} = {} & \mathrm{Lin}(d_{\text{ds}}, d_{\text{flat}}, N, b) \\ & + \mathrm{Smooth}(L, d_{\text{out}}) + \mathrm{Reduce}(L, d_{\text{out}}). \end{aligned}$$

This distinguishes the projected width per patch from the reconstructed width per token.

Patch-end. SpaceByte combines patch representations at patch ends before combining the reconstructed stream with the token-level residual:

$$C_{\text{patch-end}} = \mathrm{Reduce}(N, d_{\text{ds}}) + \mathrm{Smooth}(L, d) + \mathrm{Reduce}(L, d).$$

Attention. Hourglass with mean pooling uses Equation 14 with $L_q = L$ and $L_k = N$, plus two normalizations, smoothing, and residual reduction. Its windowed variant sets $M = \lceil LN - Lw \rceil$. Unlike copy-based reconstruction, this adds query, key, value, and output projections as well as attention pair costs.

I.7 SCOPE AND LIMITATIONS

The estimates support comparisons under a shared counting convention, with the following qualifications.

Attention pair counts. Full self-attention is charged for $\lceil L \rceil^2$ pairs, including those excluded by causal masking. This overestimates pairwise arithmetic relative to an implementation that skips causally masked pairs, especially at long sequence lengths.

Expected compression. Static estimates use assumed patch lengths rather than measuring the boundaries selected on each input. Shorter realized patches increase the nested sequence length and compute without changing the static estimate. The comparison protocol in Section 4 therefore distinguishes these estimates from realized compression and accumulated training FLOPs.

Embedding and positional encoding. Embeddings are charged as dense matrix multiplications, which materially affects the BPE reference. Conversely, the RoPE count is independent of sequence length and head count, and normalization internal to attention is not counted separately.

Auxiliary entropy loss. The entropy scorer's upsampler-side auxiliary cross-entropy is omitted from the count.

Boundary selection. Fixed thresholding and Top-K selection are assigned zero cost, and adaptive thresholding is assigned $6(\lceil L \rceil - 1)$. These conventions omit the full cost of selection and ranking, so they should not be read as runtime estimates for the boundary rules.