MULTI-LEVEL TRANSFORMER: A MINIMAL TRANSFORMER EXTENSION FOR LEARNED SEQUENCE CHUNKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Byte-level sequence models waste significant capacity on the inherent compressibility of raw data. Modern tokenizers address this by chunking byte sequences into tokens with roughly uniform information density, a heuristic central to scaling large language models. However, tokenizers are static, trained once, and blind to the downstream model. The Multi-Level Transformer (MLT) introduces a minimal modification to standard Transformers, augmenting them with hierarchical processing at different temporal resolutions controlled by a target compression rate. Unlike prior dynamic-chunking approaches that rely on ratio-balancing objectives or boundary heuristics, MLT enforces capacity by construction and recovers a standard dense Transformer at full rate. Applied to byte-level language modeling, MLT outperforms prior chunking mechanisms under matched compute and matches a tokenized Transformer trained on the same data, learning compression rates that reflect the differing information densities of English, Chinese, and DNA sequences. The discovered chunk boundaries closely match those of trained tokenizers on both English and Chinese, emerging purely from the language-modeling objective. Applied to token-level language modeling, MLT learns to skip 20–30% of tokens while maintaining baseline performance, yielding significant inference savings. Finally, MLT outperforms natural language tokenizers on DNA sequence modeling, demonstrating the flexibility of learned chunking beyond language domains.

1 INTRODUCTION

Subword tokenizers such as BPE (Sennrich et al., 2016) and SentencePiece (Kudo, 2018) compress raw byte sequences into shorter token sequences. Their core function is to smooth information density so that Transformers process units of roughly uniform semantic weight (Erdogan et al., 2026; Lester et al., 2024). Without this smoothing, a model must allocate heavy computation universally just to handle occasional dense, unpredictable segments, wasting vast resources on highly predictable ones. Naive fixed-step strategies such as merging every four bytes, or random chunking, suffer from the same problem, since they implicitly assume that every position in a sequence is equally informative, which is rarely the case.

While tokenizers solve this density problem, they are static. They are trained once on a fixed corpus and applied identically regardless of the downstream model, task, or domain. A BPE tokenizer trained on English may aggressively fragment Chinese or code (Petrov et al., 2023); one trained on text performs poorly on DNA sequences. More fundamentally, a fixed vocabulary cannot adapt to the model it serves. A pattern that is unpredictable to a small model may be trivial for a larger one, yet both receive the same chunked input. As models scale, the optimal compression level changes (Lester et al., 2024), but the tokenizer remains frozen.

What we want instead is a system that chunks its input into segments of roughly equal information content, adapting to both the data and the model. Such a system would prepare inputs for higher-level processing by concentrating compute on the most informative segments, letting predictable stretches pass through cheaply. Hierarchical processing provides this. A lightweight lower level sees every position; a deeper, more expensive upper level is invoked only where the lower level determines it is needed. The result is reduced inference cost with minimal or no loss in model performance. As we

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.

show in our experiments, moderate compression can even *improve* downstream performance, likely because curating what enters the deeper model's context sharpens its attention. The approach applies at multiple levels of the input representation, whether bytes or tokens.

This idea has roots in neuroscience. Research on event segmentation (Zacks & Tversky, 2001; Radvansky & Zacks, 2017) shows that the brain parses experience into discrete, hierarchical events rather than processing it as a uniform stream. Lower-level sensory regions detect fine-grained events, and these compose upward through the cortical hierarchy, where signals from different modalities are fused and events become progressively more abstract (Baldassano et al., 2017). The boundaries between events emerge from the tasks being solved, not from any single sensory cue (Shim et al., 2024). We do not model these biological mechanisms directly, but the principle of hierarchical, task-driven segmentation is what motivates the architecture we propose (Appendix L).

The "Bitter Lesson" (Sutton, 2019; Perić, 2025) suggests that general methods leveraging computation will eventually outperform hand-designed heuristics. Just as learned features replaced hand-crafted filters in computer vision, the static tokenizer should give way to a learned, dynamic chunker. The modification required is minimal, which preserves compatibility with the existing Transformer ecosystem of tooling, hardware kernels, and scaling recipes. We call the resulting architecture the **Multi-Level Transformer (MLT)**. Compared with the closest prior work, H-Net (Hwang et al., 2025), MLT needs no ratio loss, similarity-based boundary detector, smoothing, or inter-stage normalization, and it enforces the compression rate exactly by construction (Section 2). Our contributions include:

- **Architecture.** By adding routing and gather/scatter operations to a standard Transformer, we obtain hierarchical processing at multiple temporal resolutions controlled by a single parameter ρ. The architecture reduces exactly to a dense Transformer at $\rho = 1$. In our experiments we focus on the two-level case.
- **Optimization.** The straight-through estimator on binary routing decisions is a first-order approximation of the REINFORCE gradient, and the encoder residual is a natural counterfactual baseline (Appendix A).
- **Byte-level modeling.** Under matched compute, MLT outperforms controlled re-implementations of prior chunking mechanisms from the Medium (205M) scale upward and matches SP-32k at equal training data, and its preferred compression rate adapts to the data: one Level-2 position per $\sim$4.5 bytes on English, $\sim$3.3 on Chinese, and 6 on DNA. The chunk boundaries closely match those of trained tokenizers on both English and Chinese, emerging purely from the language-modeling objective. On DNA sequences, MLT outperforms models relying on language-trained tokenizers.
- **Token-level modeling.** When applied to tokens as inputs, mimicking nested hierarchical processing, MLT skips 20–30% of tokens while maintaining baseline performance, reducing per-token inference FLOPs and the Level-2 KV cache accordingly.

2 Related Work

2.1 Dynamic Chunking and Hierarchical Models

Tokenizer-free models such as Charformer (Tay et al., 2021), CANINE (Clark et al., 2022), and ByT5 (Xue et al., 2022) operate directly on characters or bytes, removing the static tokenizer entirely. However, because they process every position at full depth, they incur significant computational costs, motivating the move toward learned segmentation.

Hierarchical temporal abstraction offers a way to reduce this cost. The idea of chunking sequences based on information content dates back to Schmidhuber's principle of history compression (Schmidhuber, 1992), where only unpredictable inputs are passed to higher levels of a recurrent hierarchy. Early work such as the Clockwork RNN (Koutnik et al., 2014) segmented time into fixed intervals, but fixed-step approaches cannot capture variable information density.

More recent architectures move toward learned segmentation. The Hourglass Transformer (Nawrot et al., 2022) introduced static pooling and unpooling at predetermined positions, and SpaceByte (Slagle, 2024) uses whitespace heuristics that work on English but fail where whitespace is uninformative. Nawrot et al. (2023) extends pooling by learning boundary positions through a differentiable softmax-

based selection mechanism. H-Net (Hwang et al., 2025) takes a further step, using a U-Net (Ronneberger et al., 2015) architecture with cosine-similarity routing to select boundaries end-to-end. Dauncey & Wattenhofer (2026) trains boundary selection with REINFORCE, and the Byte Latent Transformer (Pagnoni et al., 2024) groups bytes into patches of roughly equal entropy, demonstrating strong scaling properties. ByteFlow (Deng et al., 2026) also fixes the number of promoted positions with top-K, but ranks them by a parameter-free coding-rate criterion instead of a learned router, and selects over the full sequence during training. These methods introduce various heuristics and auxiliary losses to enable and stabilize dynamic boundary selection. MLT replaces these training-time heuristics with a single auxiliary loss for causal inference (Section 3.4), framing compression purely as a conditional routing problem.

Relationship to H-Net. H-Net is the closest prior work: both methods learn boundaries end-to-end within an encoder, inner network, and decoder hierarchy. MLT differs in four mechanisms. (i) The compression rate is enforced exactly by expert-choice top-k rather than encouraged by a ratio loss. (ii) Boundaries are scored by an unconstrained MLP rather than by cosine similarity between adjacent embeddings. (iii) Level-2 outputs are scattered back to the positions they came from instead of being smoothed by an EMA and broadcast to all positions. (iv) MLT uses no inter-stage normalization or zero-initialized residual projection. Our H-Net EC baseline keeps (ii)–(iv) but shares MLT's top-k routing (Appendix E), so it isolates the chunking mechanisms from the capacity mechanism; Appendix G ablates them one at a time.

2.2 Conditional Computation and Routing

The idea that not all inputs require the same amount of computation is central to conditional computation (Bengio et al., 2013). In the context of Transformers, this has been explored through early-exit methods that terminate computation for easy tokens, and through mixture-of-experts (MoE) architectures that route tokens to specialized sub-networks. Expert-Choice routing (Zhou et al., 2022) avoids the auxiliary balancing losses typical of MoE by inverting the selection so that each expert picks its top-k tokens. Mixture-of-Depths (MoD) (Raposo et al., 2024) applies this primitive to skip computation on a per-layer basis, selectively dropping tokens as they propagate through the network. MLT uses the same top-k routing primitive but applies it at a macro-architectural level. Rather than making independent per-layer skip decisions, it makes a single routing decision before a deep inner block, channeling only selected representations into it. Because one decision gates the entire inner block, the selected positions form a single compressed sequence shared by all Level-2 layers, which is what allows the routing decisions to be read as chunk boundaries (Section 4.3).

Token reduction is also studied in vision. Hierarchical backbones downsample spatial resolution at fixed stages (Ronneberger et al., 2015; Liu et al., 2021), and token pruning or merging methods remove or combine redundant patches inside Vision Transformers, either with learned scores at fixed keep ratios (Rao et al., 2021) or by similarity-based matching (Bolya et al., 2023). These methods target bidirectional encoders and discard or merge tokens, whereas MLT makes a causal selection for autoregressive modeling and keeps every position in Level 1.

3 Methodology

3.1 The Multi-Level Transformer

Dynamic chunking is, at its core, hierarchical modeling. A small, fast model processes every position in a sequence, and a larger, more expensive model is invoked only when needed. A router decides, at each time-step, whether the current position is salient enough to warrant the larger model's attention. This framing admits multiple complementary interpretations. As dynamic chunking, the router partitions the input into variable-length segments. As compression, the larger model operates on a temporally abstract context whose keys and values correspond only to selected positions. As adaptive computation, the larger model's queries are temporally sparse, concentrating compute on the most informative moments.

These three perspectives unify under a single mechanism, the routing score σ_t, which plays a dual role. On the query side, it determines *when* to invoke the deep model, controlling adaptive computation. On the key/value side, it determines *what* is stored in context, controlling compression. Transformer

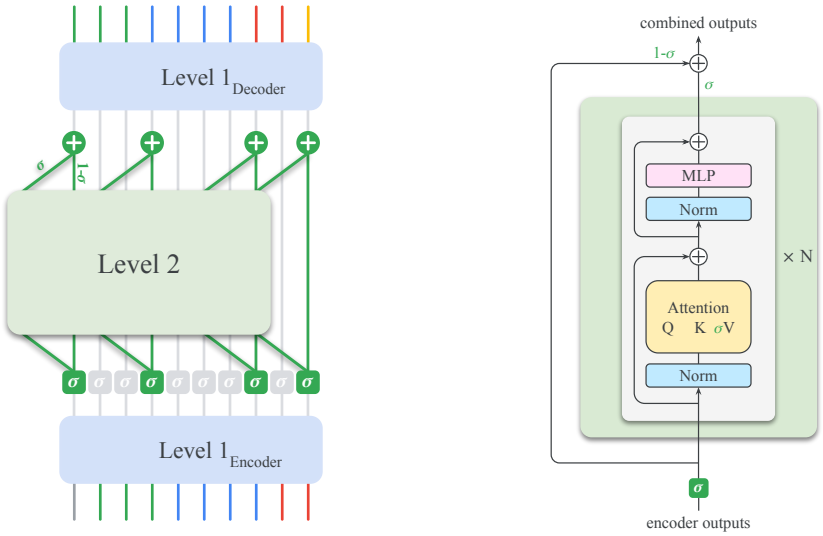

Figure 1: **Left:** MLT architecture. Level 1 (encoder and decoder) runs at full sequence resolution; Level 2 processes only the ρ-fraction of positions selected by the router. The merge formula weights Level 2 output against the encoder residual using the routing score σ_t. **Right:** Level 2 attention block used in all experiments, with values weighted by σ.

attention naturally decomposes into these two complementary effects, making the architecture well-suited for learned chunking.

Following the Hourglass Transformer (Nawrot et al., 2022) and H-Net (Hwang et al., 2025), hierarchical sequence processing requires four components: an encoder over the full input, a router that selects positions for deeper computation, a higher-level module operating on the selected subset, and a decoder that combines both levels to model the raw input. Unlike prior work, we replace dedicated chunker and dechunker modules with simple gather and scatter operations (Section 3.2; ablated against broadcast de-chunking in Appendix G). In our view, the small model is Level 1 (encoder + decoder) and the large model is Level 1 + Level 2.

Concretely, a two-level MLT (Figure 1) consists of four components ($f_{1,E}$, R, f_2, $f_{1,D}$):

- Level 1 Encoder ($f_{1,E}$). Processes the full input sequence to produce embeddings $h^E_{1:T}$.
- Router (R). Computes a score $\sigma_t \in [0, 1]$ for each encoded embedding.
- Level 2 (f_2). Operates only on the $\lceil \rho T \rceil$ embeddings selected by the router, where $\rho \in (0, 1]$ is the target compression rate. May be any sequence-to-sequence model, including another MLT. Level 2 typically handles the bulk of the total computation.
- Level 1 Decoder ($f_{1,D}$). Processes the combined Level1 Encoder and Level 2 outputs to produce outputs corresponding to the full input sequence.

The total layer budget is $n = n_{1,E} + n_2 + n_{1,D}$. In our experiments, we set $n_{1,E} = n_{1,D} = 2$ and allocate the remaining layers to Level 2 for fair comparison at matched total depth (we investigate the effect of this split in Appendix H). The routing score σ_t enters Level 2 computation in two ways. On the query pathway, the merge of Level-2 and Level-1 encoder outputs (Eq. 2) incorporates σ_t into the representations that Level 2 attends from. On the key/value pathway, following the observation of Lou et al. (2024) that sparse attention benefits from weighting attended positions by their selection confidence, we multiply values in every Level-2 attention layer by σ_t (Figure 1, right; ablated in Appendix I). This provides a small but consistent improvement across model sizes. The optimal ρ is primarily data-dependent and stable across model sizes (Section 4.1); a single sweep with a small MLT suffices to identify it.

When $\rho = 1$, the top-k operator selects every position, the gather/scatter reduces to the identity, and the architecture becomes structurally equivalent to a dense Transformer of depth $n_{1,E} + n_2 + n_{1,D}$.

MLT is therefore a structural superset of a standard Transformer. The same model class smoothly interpolates between dense computation and aggressive compression as ρ decreases.

3.2 Hard Top-k Routing

We select which positions Level 2 processes using *expert-choice* top-k selection (Zhou et al., 2022; Raposo et al., 2024). The router produces a scalar score σ_t at each position (Section 3.3); Level 2 processes the $k = \lceil \rho \cdot T \rceil$ highest-scoring positions out of the length-T encoder outputs $h^E_{1:T}$:

$$\mathcal{I} \;=\; \mathrm{TopK}(\sigma_{1:T},\; k). \tag{1}$$

We then *gather* the selected positions into a length-k sequence, run Level 2 on it, and *scatter* the outputs back to their original time-steps:

$$h^{D_{\text{in}}}_t = \begin{cases} \sigma_t \cdot f_2(h^E)_t + (1-\sigma_t)\cdot h^E_t & t \in \mathcal{I} \\ h^E_t & t \notin \mathcal{I}. \end{cases} \tag{2}$$

Unlike the Hourglass Transformer and H-Net, which broadcast compressed representations back to all positions and introduce mechanisms like smoothing and normalization to stabilize training, MLT scatters each Level-2 output to the same time-step it was gathered from. Non-selected positions retain the encoder residual h^E_t unchanged. This mirrors the biological picture more faithfully than broadcasting, since in event segmentation higher-level processing occurs at specific boundary moments rather than being smeared across the entire stream. This keeps gradient flow through Level 2 sparse and well-localized (we ablate scatter against broadcast in Appendix G). The encoder residual is also a natural counterfactual baseline for the routing decision. Because both paths are evaluated at every routed position (the encoder residual for "do not route" and the Level-2 output for "route"), the straight-through estimator has access to an exact counterfactual rather than a learned baseline, which keeps the bias of the gradient approximation small (formalized in Appendix A).

The compression rate is fixed to exactly ρ by construction. The realized fraction of routed positions is always $\lceil \rho T \rceil / T$, so no auxiliary ratio loss or balancing term is needed. Because top-k over the full sequence depends on all T scores, the routing decision becomes non-causal. At inference we therefore train a small *causal router* to predict the top-k decision from causal context alone (Section 3.4).

3.3 Router Architecture

We use a simple single-hidden-layer MLP as the routing function, applied independently to each encoder embedding. This is a deliberate choice over more structured alternatives such as cosine similarity between adjacent embeddings (as in H-Net). The MLP imposes no functional form on what constitutes a boundary, allowing the model to learn arbitrary routing patterns from the task objective. The router produces a scalar score:

$$\sigma_t \;=\; \mathrm{Sigmoid}\big(\mathrm{MLP}(h^E_t)\big). \tag{3}$$

In the forward pass the hard top-k membership determines which positions enter Level 2; in the backward pass gradients flow through the continuous score σ_t via a straight-through estimator; σ_t is also the merge weight in Eq. 2. We found that this simple STE works well in practice and that more elaborate alternatives such as Gumbel-Softmax sampling or concrete relaxations negatively impacted performance. We analyze the optimization properties of this formulation in Appendix A, showing why the encoder residual removes the need for the auxiliary mechanisms used in prior work. Algorithm 1 (Appendix B) summarizes the forward pass.

3.4 Causal Routing for Autoregressive Inference

Top-k over a length-T sequence is non-causal because the selection $b_t = \mathbf{1}[t \in \mathcal{I}]$ depends on all T scores. At inference we replace it with a small *causal router* r_ϕ, a one- or two-layer causal Transformer head over the encoder stream, trained alongside the main objective with a bidirectional loss:

$$\mathcal{L}_{\text{causal}} \;=\; \mathrm{BCE}\big(r_\phi(h^E),\; b\big) \;+\; \lambda \cdot \mathrm{BCE}\big(\mathrm{sg}\big(r_\phi(h^E)\big),\; \sigma\big), \tag{4}$$

where sg denotes stop-gradient. The first term distills the non-causal top-k decision into the causal predictor, similar to MoD (Raposo et al., 2024). The second term regularizes the non-causal scores σ

to remain predictable from causal context, limiting the degree of non-causality the router can exploit during training. We use $\lambda = 0.05$ across all experiments; the backward loss is not strictly necessary but has a slight stabilizing effect (ablated in Appendix I).

Unlike the non-causal router, whose scores σ directly shape the representations through the merge formula (Eq. 2), the causal router is detached from the main computation graph and does not influence the learned representations. In practice it agrees with the non-causal top-k selection on more than 99% of positions, in line with the accuracy Raposo et al. (2024) report for an analogous predictor, so the difference at inference is negligible. We ignore the causal router's small FLOP cost in the cost analysis (Appendix D). All downstream evaluations use the causal router at inference.

4 Experiments

We train MLTs at four scales, ranging from 85M to 1.1B body parameters, using modern Transformer components and the scaling proxy of Wortsman et al. (2023). Every model is trained for 100k steps (except the data-matched reference in Section 4.2) at batch size 256 on C4/en (C4/zh and HG38 in Section 4.4). All experiments are FLOP-matched against a dense SentencePiece-32k (hereafter SP-32k) Transformer at sequence length $T_0 = 512$, the reference budget of the scaling proxy. We match FLOPs rather than sequence length because a byte-level model at equal length would receive several times the compute of its tokenized counterpart. The SP-32k reference therefore sees ≈13.1B tokens (≈59B bytes), while a FLOP-matched byte-level Large MLT processes sequences of $T_1 = 1479$ bytes, ≈38B bytes in total (Appendix B.3). We report bits per byte (BPB) on held-out data and four downstream evaluations: ARC-Easy, HellaSwag, and PIQA (scoring-based) and LAMBADA (sampling-based). We found that scoring-based benchmarks are highly correlated with each other, as are sampling-based ones, so the main-text tables and the ablations report one of each, HellaSwag and LAMBADA, and Appendix C reports all four. Full architecture, optimizer, and FLOP-matching details are in Appendix B.

Our baselines are: a dense SP-32k Transformer at ρ=1 as the tokenized reference; Random routing (uniform top-k in the MLT skeleton); Space adapted (whitespace-based boundaries in the MLT skeleton); ByteFlow adapted (top-k by the parameter-free coding-rate score of Deng et al. (2026) in the MLT skeleton, Appendix F); and H-Net EC, a controlled re-implementation of H-Net that uses MLT's network skeleton and expert-choice routing but retains H-Net's core chunking mechanisms (Appendix E, Appendix G). All byte-level baselines use MLT's optimal $\rho = 0.225$ (Section 4.1), which coincides with the average compression of the SP-32k tokenizer (∼4.5 bytes per token) and is therefore a natural operating point for any method that replaces it; we did not sweep ρ separately for each baseline (Section 5).

4.1 Bytes: Choosing the Compression Rate

We first apply MLT directly to raw bytes on C4/en and sweep the routing proportion ρ at four model sizes. Table 1 shows the full sweep. Across all scales the optimum is consistently $\rho \in [0.2, 0.225]$, with a learned Level-2 that processes one in every ~ 4.5 bytes of input, which matches the average bytes-per-token of the SP-32k baseline on the same data (Appendix K). MLT discovers this rate without any tokenizer supervision, purely from the language-modeling objective. Within this range, larger models prefer slightly more aggressive compression ($\rho = 0.2$ at Large and XL vs. $\rho = 0.225$ at Small and Medium), consistent with the observation that larger models can push compression further (Lester et al., 2024). The optimal ρ is thus primarily data-dependent, with a secondary model-capacity component. All compressed configurations ($\rho < 1$) outperform the uncompressed baseline ($\rho = 1$) at every scale, suggesting that compression itself can be beneficial for downstream performance. We discuss possible explanations in Section 5. We fix $\rho = 0.225$ for all subsequent byte-level experiments.

4.2 Bytes: Comparison to Baselines and the Tokenized Reference

At fixed $\rho = 0.225$ we compare MLT against Random, Space (adapted), ByteFlow (adapted), and H-Net EC. All five byte-level models are FLOP-matched to a dense SP-32k Transformer, which we report as a tokenized reference (Table 2; Table 6 in Appendix C adds ARC-Easy and PIQA). Three observations:

Table 1: Byte-level ρ sweep on C4/en. Columns are bits per byte (BPB, ↓) on held-out data and HellaSwag (HS) and LAMBADA (LM) accuracy (↑). The optimum is consistently in $[0.2, 0.225]$ across model sizes, corresponding to ~ 4.5 bytes per chunk, matching the average bytes-per-token of modern subword tokenizers.

	SMALL			MEDIUM			LARGE			XL		
ρ	BPB	HS	LM	BPB	HS	LM	BPB	HS	LM	BPB	HS	LM
1.000	1.212	32.1	25.7	1.160	34.7	29.9	1.101	37.8	32.4	1.089	40.7	34.6
0.333	1.170	33.1	27.1	1.115	36.8	32.1	1.043	42.3	36.2	1.021	47.0	39.3
0.250	1.161	33.2	27.0	1.108	36.9	33.1	1.034	42.8	36.8	1.014	47.5	40.0
0.225	**1.149**	**33.7**	**28.1**	**1.094**	**37.9**	**33.7**	1.025	43.2	37.2	1.006	**48.1**	40.7
0.200	1.153	33.5	27.3	1.097	37.7	33.4	**1.022**	**43.4**	**37.4**	**1.004**	**48.1**	**40.9**
0.166	1.166	33.0	26.1	1.106	37.3	33.3	1.033	42.7	36.8	1.019	46.0	37.2

(i) When both see the same data, MLT matches the SP-32k reference. The DATA-MATCHED reference keeps per-step FLOPs matched and is trained on the same number of bytes as the byte-level models, and therefore uses less total compute (Appendix B.3). Under matched *total* FLOPs, SP-32k trains on 1.5–1.8× more bytes and remains ahead. H-Net (Hwang et al., 2025) matches data and total compute at once by adjusting depth so that FLOPs per byte are equal; our two references bracket that setting. **(ii)** MLT strictly dominates Random, Space (adapted), and ByteFlow (adapted) at every size and on every metric; ByteFlow's fixed coding-rate criterion improves on whitespace boundaries but trails both learned routers. **(iii)** Against H-Net EC, which differs from MLT only in the chunking mechanism (Appendix G), MLT is on par at Small (tied on HellaSwag; H-Net EC slightly ahead on LAMBADA, BPB, and PIQA) and leads on all five metrics from Medium upward, and the margin grows with size. H-Net's cosine-similarity policy is a useful inductive bias at small capacity, but at larger scales this fixed functional form appears more limiting than a learned MLP.

Table 2: Byte-level comparison at $\rho = 0.225$ against Random, Space (adapted), ByteFlow (adapted), and H-Net EC, all FLOP-matched to a dense SP-32k Transformer (token-level reference, first row). DATA-MATCHED: SP-32k at matched per-step FLOPs trained on the same number of bytes (56k–68k steps). Columns are bits per byte (BPB, ↓) on held-out C4/en and HellaSwag (HS) and LAMBADA (LM) accuracy (↑); Table 6 (Appendix C) adds ARC-Easy and PIQA. Bold marks the best result per column among byte-level methods.

	SMALL			MEDIUM			LARGE			XL		
METHOD	BPB	HS	LM	BPB	HS	LM	BPB	HS	LM	BPB	HS	LM
SP-32K	1.068	34.0	31.3	1.020	38.5	35.3	0.983	44.7	40.6	0.952	49.2	43.9
DATA-MATCHED	1.145	33.6	28.4	1.106	37.2	32.3	1.021	42.7	37.4	1.004	48.2	40.4
RANDOM	1.226	30.9	21.7	1.171	34.0	25.8	1.133	36.9	30.6	1.083	41.5	34.1
SPACE	1.193	32.4	23.3	1.143	35.2	28.0	1.086	39.7	32.1	1.047	44.7	35.2
BYTEFLOW	1.165	33.1	26.3	1.121	36.1	30.7	1.064	40.9	33.7	1.032	45.4	36.8
H-NET EC	**1.148**	**33.7**	**28.4**	1.105	37.0	32.4	1.050	42.3	35.5	1.019	46.8	38.6
MLT	1.149	**33.7**	28.1	**1.094**	**37.9**	**33.7**	**1.025**	**43.2**	**37.2**	**1.006**	**48.1**	**40.7**

4.3 BYTES: CHUNK-BOUNDARY ALIGNMENT WITH SUBWORD TOKENIZERS

Section 4.1 showed that the optimal ρ matches the average compression rate of static tokenizers. We next test whether the *positions* the router selects are also where a static tokenizer would place a boundary. We view this as an analysis of what the router learns rather than as a target: MLT receives no tokenizer supervision, so agreement indicates how much of a tokenizer's structure the language-modeling objective recovers on its own. We treat the start of each SP-32k token as a binary boundary event and compute accuracy and F1 against MLT's routing decisions at matched compute (Table 3). On English, MLT reaches 83.6 F1. H-Net EC is close behind, consistent with its cosine-similarity policy also discovering linguistically meaningful boundaries. Random routing

Table 3: Boundary alignment with SP-32k: accuracy and F1 of routing decisions against SP-32k token starts ($\rho = 0.225$ English, 0.3 Chinese).

Model	English		Chinese	
	Acc.	F1	Acc.	F1
MLT	**92.8**	**83.6**	**90.4**	**83.8**
H-Net EC	92.1	82.3	88.4	80.5
Space	90.8	79.8	–	–
Random	64.3	22.1	58.5	30.3

Table 4: Cross-domain BPB (↓), Large, FLOP-matched. The Chinese SP-32k tokenizer is trained on C4/zh.

Method	C4/zh	HG38
	ρ=0.3	ρ=1/6
MLT	**0.821**	**1.361**
H-Net EC	0.834	1.368
Random	0.934	1.412
SP-32k dense	–	1.399

reaches only 22.1 F1, close to the chance level at this rate, confirming that the alignment is not a trivial consequence of the compression rate. Space boundaries are a good prior for English, reflected in the relatively high F1 of the Space baseline. The same pattern holds on Chinese, where Space is not applicable due to the absence of whitespace. Annotated chunk-boundary examples are in Appendix J.

4.4 Cross-Domain: Chinese and HG38

We evaluate MLT on raw bytes in two non-English settings to test whether the same architecture transfers without modification. If dynamic chunking adapts to the data, the optimal ρ should change across domains to reflect their different information densities. We report bits per byte (BPB), the standard metric for byte-level language modeling, on held-out evaluation data (Table 4).

Chinese (C4/zh). Chinese characters are encoded as 3 UTF-8 bytes each, but C4/zh is a web-crawled dataset that also contains English text, numbers, URLs, and HTML. A SP-32k tokenizer trained on C4/zh compresses the ASCII portions efficiently while Chinese characters remain close to 3 bytes per token, yielding an average of $\sim$3.3 bytes per token across the mix. We sweep ρ at matched compute and find the optimum at $\rho = 0.3$, consistent with this higher bytes-per-token ratio. MLT outperforms both H-Net EC and Random at this setting.

HG38 (DNA). The human genome consists of a 4-letter alphabet (A, C, G, T) with no whitespace or punctuation. A language-trained SP-32k tokenizer is a poor fit for this data. We sweep ρ at matched compute and find the optimum at $\rho = 1/6$. MLT outperforms both the SP-32k tokenized baseline and H-Net EC, so learned chunking transfers to domains where static tokenizers have no useful prior. Across the three datasets, the optimal ρ varies substantially (0.225 for English, 0.3 for Chinese, $1/6$ for DNA), following the information structure of each domain.

4.5 Tokens-as-Input: A FLOP-Matched ρ Sweep

The byte-level experiments use MLT as a replacement for static tokenization. This experiment applies MLT to already-tokenized inputs, testing whether it can identify and skip redundant tokens. This setting approximates a three-level hierarchy (bytes $\rightarrow$ tokens $\rightarrow$ routed tokens), with the tokenizer acting as a fixed first level of compression. We run MLT on SP-32k tokens and sweep $\rho \in [0.1, 1.0]$ at matched training FLOPs (Figure 2). At $\rho = 0.8$ the Large model *improves* over the dense baseline on HellaSwag (45.1 vs. 44.7) and matches it on LAMBADA, while routing only 80% of tokens through Level 2. Performance remains at or above baseline down to $\rho \approx 0.7$; below $\rho \approx 0.5$ it degrades gradually rather than catastrophically.

Roughly 20–30$\%$ of tokens in a strong SentencePiece baseline are therefore redundant for next-token prediction, and a learned chunker can identify and skip them at no quality cost. Because Level 2 is the dominant cost component, this translates into inference savings on tokenized pipelines without changing the input, the loss, or the tokenizer. For the Large model, per-token decoding FLOPs outside attention drop by $\approx$15% at $\rho = 0.8$ and $\approx$23% at $\rho = 0.7$, while Level-2 attention FLOPs and KV cache shrink as ρ^2 and ρ (Appendix D).

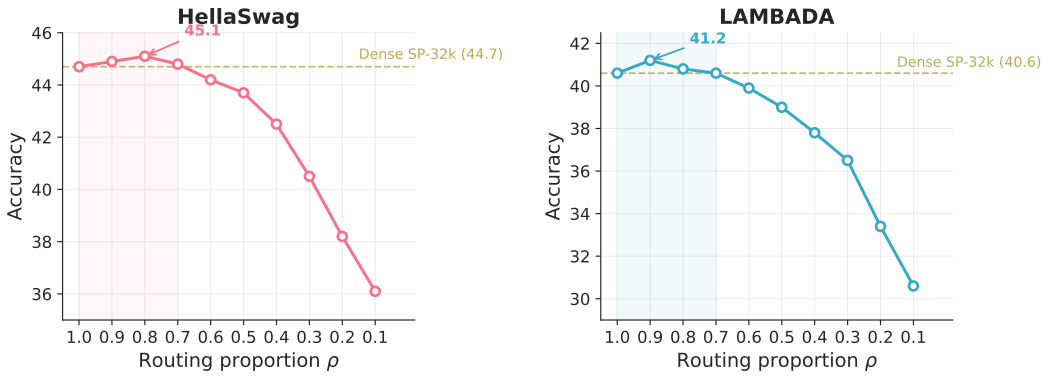

Figure 2: Token-as-input ρ sweep on C4/en (Large, FLOP-matched). The dashed line marks the dense SP-32k baseline (ρ=1). MLT matches or exceeds the baseline for $\rho \geq 0.7$, peaking at $\rho = 0.8$ on HellaSwag and $\rho = 0.9$ on LAMBADA. The shaded region marks ρ values where MLT outperforms the dense baseline.

5 Discussion

System 1 vs. System 2. MLT implements a dual-process view of computation (Kahneman, 2011). Level 1 acts as System 1: fast, reactive, always active over every position. Level 2 acts as System 2: slow, expensive, engaged only when the router deems the current position salient. The neuroscience parallel, in which lower sensory streams process every observation while higher cognitive levels engage selectively at event boundaries, motivates the architectural choice of a single hierarchical routing decision rather than independent per-layer skipping.

Why compression can improve performance. The ρ-sweep on both bytes and tokens shows a consistent pattern: performance initially increases with compression before eventually degrading. This aligns with recent findings on attention dilution. As the number of keys grows, softmax attention disperses weight increasingly uniformly, making it harder for the model to attend sharply to informative positions (Velickovic et al., 2024; Nakanishi, 2025). By curating which positions enter Level 2's context, the router implicitly sharpens the attention distribution. Recent work on gated attention (Qiu et al., 2025) provides independent evidence that learned sparsity over attention inputs improves performance and eliminates pathological behaviors such as attention sinks.

Recursive and multi-stream extensions. Level 2 may itself be an MLT, so deeper hierarchies follow naturally. The tokens-as-input results (Section 4.5) are a first instance: above a tokenizer acting as a fixed first level, a learned level still skips 20–30% of positions at baseline performance, so further compression remains available beyond the first stage. Training deep recursive MLTs end-to-end is left for future work. The same construction extends to *multimodal fusion*: each modality gets its own Level 1 and ρ feeding a shared Level 2, so redundant streams such as audio and video can be compressed aggressively.

Limitations. We train only two-level MLTs end-to-end; the tokens-as-input results suggest deeper hierarchies are promising, but we leave them to future work. On English, byte-level MLT matches SP-32k at equal training bytes but remains behind it at matched total FLOPs. Byte-level baselines use MLT's optimal ρ rather than individually tuned rates, and H-Net EC replaces H-Net's thresholded boundaries and ratio loss with expert-choice routing. Inference savings are FLOP and KV-cache estimates, not wall-clock measurements. Finally, the causal router approximates the non-causal top-k selection at inference (>99% agreement), and a fixed per-sequence ρ assumes stable information density (Appendix K).

Reproducibility Statement

Model configurations are listed in Table 5, and the optimizer, schedule, and training budget in Appendix B. The FLOP-matching procedure and the matched sequence lengths are given in Appen-

dices B.3 and D, and the H-Net EC baseline is specified in Appendix E. All datasets (C4/en, C4/zh, and the HG38 human reference genome) are publicly available. We do not release code; Algorithm 1 and Section 3 specify the full modification to a standard Transformer.

AI USE STATEMENT

In this work, we used generative AI tools to draft and edit parts of the manuscript, to search for and verify related work and references, to write plotting code for figures, and to format tables, references, and the conference template. We have not used generative AI tools to generate datasets, implement the method, or run experiments. We have reviewed all AI-assisted work, and we take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, et al. The deepmind jax ecosystem. *GitHub*, 2020.

Christopher Baldassano, Janice Chen, Asieh Zadbood, Jonathan W Pillow, Uri Hasson, and Kenneth A Norman. Discovering event structure in continuous narrative perception and memory. *Neuron*, 95 (3):709–721, 2017.

Aya Ben-Yakov and Yadin Dudai. Constructing realistic engrams: poststimulus activity of hippocampus and dorsal striatum predicts subsequent episodic memory. *Journal of Neuroscience*, 31(24): 9032–9042, 2011.

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

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations*, 2023.

Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022.

Sam Dauncey and Roger Wattenhofer. You can learn tokenization end-to-end with reinforcement learning. *arXiv preprint arXiv:2602.13940*, 2026.

Chunyuan Deng, Sanket Lokegaonkar, Colin Lockard, Besnik Fetahu, Nasser Zalmout, and Xian Li. Byteflow: Language modeling through adaptive byte compression without a tokenizer. *arXiv preprint arXiv:2603.03583*, 2026.

Lukas Edman and Alexander Fraser. Toppling the hierarchy in byte-level language modeling. *arXiv preprint arXiv:2609.00463*, 2026.

Mete Erdogan, Abhiram Gorle, Shubham Chandak, Mert Pilanci, and Tsachy Weissman. An information-theoretic perspective on LLM tokenizers. *arXiv preprint arXiv:2601.09039*, 2026.

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

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. In *International conference on machine learning*, pp. 1863–1871. PMLR, 2014.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*, 2018.

Brian Lester, Jaehoon Lee, Alex Alemi, Jeffrey Pennington, Adam Roberts, Jascha Sohl-Dickstein, and Noah Constant. Training LLMs over neurally compressed text. *arXiv preprint arXiv:2404.03626*, 2024.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 9992–10002. Ieee, 2021.

Chao Lou, Zixia Jia, Zilong Zheng, and Kewei Tu. Sparser is faster and less is more: Efficient sparse attention for long-range transformers. *arXiv preprint arXiv:2406.16747*, 2024.

Sebastian Michelmann, Uri Hasson, and Kenneth A Norman. Evidence that event boundaries are access points for memory retrieval. *Psychological Science*, 34(3):326–344, 2023.

Ken M. Nakanishi. Scalable-softmax is superior for attention. *arXiv preprint arXiv:2501.19399*, 2025.

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 the Association for Computational Linguistics: NAACL 2022*, pp. 1559–1571, 2022.

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

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte latent transformer: Patches scale better than tokens. *arXiv preprint arXiv:2412.09871*, 2024.

Luca Perić. The bitter lesson is coming for tokenization. `https://lucalp.dev/bitter-lessontokenization-and-blt`, June 2025.

Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. Language model tokenizers introduce unfairness between languages. *Advances in neural information processing systems*, 36: 36963–36990, 2023.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

Gabriel A Radvansky and Jeffrey M Zacks. Event boundaries in memory and cognition. *Current opinion in behavioral sciences*, 17:133–140, 2017.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 1715–1725, 2016.

Sunjae Shim, Franck B Mugisho, Lila Davachi, and Christopher Baldassano. Generating event boundaries in memory without prediction error. *bioRxiv*, 2024.

Kevin Slagle. Spacebyte: Towards deleting tokenization from large language modeling. *Advances in Neural Information Processing Systems*, 37:124925–124950, 2024.

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 13(1):38, 2019.

Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*, 2021.

Petar Velickovic, Christos Perivolaropoulos, Federico Barbero, and Razvan Pascanu. Softmax is not enough (for sharp size generalisation). *arXiv preprint arXiv:2410.01104*, 2024.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

Jeffrey M Zacks and Barbara Tversky. Event structure in perception and conception. *Psychological bulletin*, 127(1):3, 2001.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

A Optimization Dynamics of Routing

The router's goal is to select, at each position, the routing that best optimizes the task objective (in our case, negative log-likelihood). Since routing decisions are discrete, gradients do not flow through them directly.

The principled approach is REINFORCE (Williams, 1992), which provides unbiased gradients but suffers from high variance. Recent work on end-to-end boundary selection (Dauncey & Wattenhofer, 2026) shows that REINFORCE can work, but requires careful variance reduction through early-exit baselines, time discounting, and batch-relative advantages.

The straight-through estimator (STE) (Bengio et al., 2013) offers an alternative that trades a small bias for substantially lower variance. This bias is minimized when a good counterfactual is available. For binary routing decisions like ours, we can evaluate both paths. The encoder residual ("do not route") is computed at every position, and the Level 2 output ("route") is computed conditionally. This gives us an exact counterfactual rather than a learned baseline.

We formalize this below, showing that the STE gradient is a first-order approximation of the REINFORCE gradient, and that the encoder residual in Eq. 2 is what makes this approximation tight.

A.1 Problem Formulation

Let x be the input state and consider two functional pathways, $f(x)$ and $g(x)$. The routing policy is parameterized by θ, yielding a mixing weight $\sigma(x;\theta) \in [0,1]$.

Soft routing. Uses σ directly in the forward pass:

$$y_{soft} = \sigma f(x) + (1-\sigma)g(x), \qquad J_{soft}(\theta) = L(y_{soft}). \tag{5}$$

Hard routing. Samples a discrete decision $z \sim \text{Bernoulli}(\sigma)$:

$$y_{hard} = zf(x) + (1-z)g(x), \qquad J_{hard}(\theta) = \mathbb{E}_{z\sim\pi_\theta}[L(y_{hard})]. \tag{6}$$

A.2 Gradient Analysis

Soft routing. Differentiating J_{soft} directly:

$$\nabla_\theta J_{soft} = \nabla_y L(y_{soft})^T (f(x) - g(x))\, \nabla_\theta \sigma. \tag{7}$$

The gradient depends on the difference $f(x) - g(x)$ between the two pathways, scaled by the local loss landscape at y_{soft}.

Hard routing (REINFORCE). The score-function estimator for the hard objective gives:

$$\nabla_\theta J_{hard} = (L(f(x)) - L(g(x)))\, \nabla_\theta \sigma. \tag{8}$$

The term $L(f) - L(g)$ acts as the reward signal; $g(x)$ serves inherently as a baseline, centering the gradient.

Connection. A first-order Taylor expansion of the loss around the observed outcome shows that the soft-routing gradient (Eq. 7) is a local linearization of the REINFORCE gradient (Eq. 8):

Proof. Consider $z = 1$ ($y_{hard} = f(x)$). Expanding $L(g)$ around $f(x)$:

$$L(g) \approx L(f) + \nabla_y L(f)^T (g - f) \implies L(f) - L(g) \approx \nabla_y L(f)^T (f - g). \tag{9}$$

The case $z = 0$ is symmetric. In both cases, substituting into Eq. 8 recovers Eq. 7 (evaluated at the observed y rather than y_{soft}). The approximation holds when the curvature $\frac{1}{2}(f-g)^T \mathbf{H}_L (f-g)$ is small relative to the gradient magnitude. □

The STE (and by extension soft routing) therefore trades an $\mathcal{O}(\|f-g\|^2)$ bias for no additional variance from the routing decision, whereas REINFORCE is unbiased but high-variance. In practice this bias appears to be a favorable trade-off. It shrinks as the encoder residual baseline brings f and g closer (see below), and recent work confirms that the unbiased alternative becomes competitive only at scale where variance can be averaged out (Dauncey & Wattenhofer, 2026).

A.3 The Role of the Encoder Residual

Both the soft and hard gradients depend on the difference $f(x) - g(x)$. If no baseline is provided (i.e. $g(x) = 0$, corresponding to $y = \sigma f(x)$), the gradient approximates the loss landscape between $f(x)$ and the origin, introducing bias proportional to $\|f(x)\|$.

The counterfactual in MLT is concrete. At positions not routed through Level 2, the decoder receives the encoder output h_t^E unchanged. The merge in Eq. 2 encodes this directly by pairing the Level-2 output $f_2(h^E)_t$ as $f(x)$ against the encoder residual h_t^E as $g(x)$. Because $g(x)$ is itself a useful representation (the encoder has already processed the full sequence), the distance $\|f - g\|$ is small relative to $\|f\|$, keeping the Hessian correction term in the Taylor approximation negligible.

Non-selected positions must pass through the encoder residual without modification. If Level-2 output were broadcast to non-selected positions (as in H-Net's de-chunking), the baseline $g(x)$ would no longer represent the true counterfactual of "not routing," and the resulting gradient would carry a systematic bias toward the broadcast representation rather than toward the routing decision itself.

EMA smoothing across adjacent compressed vectors provides a similar baseline indirectly but introduces a recurrence. The encoder-residual construction achieves the same effect without additional components.

B Model Architectures and Training

Algorithm 1 Multi-Level Transformer

def $\mathrm{MLT}(X, \rho)$: *# X: input sequence $[T, D]$, ρ: compression rate*
  $H \leftarrow \mathrm{Encoder}(X)$ *# Level 1 Encoder $[T, D]$*
  $\sigma \leftarrow \mathrm{sigmoid}(\mathrm{Router}(H))$ *# Router scores $[T]$*
  $k \leftarrow \lceil \rho \cdot T \rceil$
  $\mathcal{I} \leftarrow \mathrm{top_k}(\sigma,\ k)$ *# Selected indices $[K]$*
  $H_{\mathcal{I}} \leftarrow \mathrm{Level2}(H[\mathcal{I}])$ *# Gather + Level 2 $[K, D]$*
  $H[\mathcal{I}] \leftarrow \sigma_{\mathcal{I}} \cdot H_{\mathcal{I}} + (1 - \sigma_{\mathcal{I}}) \cdot H[\mathcal{I}]$ *# Scatter + Merge*
  return $\mathrm{Decoder}(H)$ *# Level 1 Decoder $[T, D]$*

B.1 Architecture configurations

We train four MLT sizes following the scaling proxy of Wortsman et al. (2023). All sub-modules use modern Transformer components: SwiGLU activations, RoPE positional encoding, QK-Norm for training stability, z-loss regularization, and RMS pre-norm. Head dimension is fixed at 64; we scale the number of layers and heads jointly. SwiGLU MLP hidden width is $\frac{8}{3}d$, rounded to the nearest multiple of 128 for kernel efficiency. We use $n_{1,E} = n_{1,D} = 2$ (4 total Level-1 layers) throughout and allocate the remaining layers to Level 2; we ablate this split in Appendix H.

Table 5: MLT model configurations. Layers and heads scale jointly; head dimension is 64. MLP hidden widths are $\frac{8}{3}d$ rounded to the nearest multiple of 128. Parameter counts are body parameters (excluding embeddings and unembedding).

Size	Body params	Layers (n)	Heads	d	MLP hidden	$n_{1,E}$	n_2	$n_{1,D}$
Small	85M	12	12	768	2048	2	8	2
Medium	205M	16	16	1024	2688	2	12	2
Large	530M	22	22	1408	3712	2	18	2
XL	1.1B	28	28	1792	4736	2	24	2

B.2 Optimization

We follow the optimization procedure of Wortsman et al. (2023) with two modifications. First, we use relative weight decay of 0.1 instead of absolute weight decay of $1\mathrm{e}{-4}$, which we found more

stable across model sizes. Second, we set $\beta_2 = 0.97$ instead of 0.95. All other settings are standard: AdamW with $\beta_1 = 0.9$, gradient clipping at 1.0, $100k$ training steps at batch size 256, peak learning rate $1\text{e}{-3}$ with 5% linear warmup and cosine decay to $\sim 10\%$ of peak.

A single learning-rate schedule is used for all parameters. We do not separately tune learning rates for Level 1 vs. Level 2 modules; as observed in H-Net (Hwang et al., 2025), additional performance is likely available from per-module tuning, and we leave this for future work.

B.3 FLOP MATCHING

All experiments are FLOP-matched to a dense SP-32k Transformer at sequence length $T_0 = 512$ (the Wortsman scaling-proxy reference budget) and run for the same 100k steps, so total compute is matched; the only exception is the data-matched reference described below. We solve for the MLT Level-1 sequence length T_1 that matches the baseline's total forward FLOPs, accounting for layer split, attention cost, and the unembedding-cost difference between vocabularies. The full derivation is in Appendix D.

Data matching. FLOP matching fixes the per-step compute but not the amount of data. At equal steps, the SP-32k reference processes $T_0\,\bar{b} \approx 2304$ bytes per sequence ($\bar{b} \approx 4.5$ bytes per SP-32k token on C4/en), whereas a byte-level MLT processes only T_1 bytes, so over 100k steps the reference sees ≈59B bytes against 33–40B. For the DATA-MATCHED reference we keep the SP-32k architecture, sequence length, and batch size, and shorten training to $N_{\text{SP}} = N\,T_1/(T_0\,\bar{b})$ steps with $N = 100$k. With $T_1 = 1283$, 1362, 1479, and 1576 at $\rho = 0.225$, this gives ≈56k, 59k, 64k, and 68k steps for Small, Medium, Large, and XL. Both models then see the same number of training bytes at matched per-step FLOPs, while the reference uses proportionally less total compute. H-Net (Hwang et al., 2025) instead feeds every model the same bytes per sequence and adjusts depth to equalize FLOPs, which matches data and total compute at the same time.

B.4 IMPLEMENTATION AND COMPUTE

All models are implemented using JAX (Babuschkin et al., 2020). Small, Medium, and Large experiments use second-generation TPUs in a 4×4 chip configuration, with runtimes between 1 and 4 days depending on model size, dataset, and ρ. XL experiments use fourth-generation TPUs in a $2\times2\times1$ chip configuration, with runtimes between 2 days and 1 week. Due to the extensive number of experiments (sweeps over ρ, model sizes, datasets, baselines, and ablations), the total compute substantially exceeds what is reported for any individual run.

C FULL BYTE-LEVEL RESULTS

Table 6 extends Table 2 with the two remaining downstream evaluations, ARC-Easy and PIQA. The setup is identical: every byte-level method uses $\rho = 0.225$ and is FLOP-matched to the dense SP-32k reference (Section 4).

Figure 3 replots the ρ sweep of Table 1 as the gain over the uncompressed model ($\rho = 1$) against the number of bytes per Level-2 position. The gain from compression grows with model size and peaks near the average bytes-per-token of the SP-32k tokenizer.

D FLOP AND INFERENCE-COST ANALYSIS

We collect the compute formulas referenced in the experiments section. The conventions are: hidden width d, FFN ratio set so the SwiGLU MLP matches the parameter count of a $4d$-wide standard MLP (i.e., FFN hidden width $d_{\text{ff}} = \frac{8}{3}d$), causal self-attention, and we count multiply–add as 2 FLOPs.

D.1 PER-LAYER FORWARD FLOPS

For a single Transformer block applied to a sequence of length T:

- **QKV projections:** $3 \cdot 2d^2 \cdot T = 6Td^2$ FLOPs.

Table 6: Full byte-level comparison at $\rho = 0.225$ (extends Table 2). All byte-level methods are FLOP-matched to a dense SP-32k Transformer (token-level reference, first row of each block); DATA-MATCHED is SP-32k at matched per-step FLOPs, trained on the same number of bytes as the byte-level models (Appendix B.3). BPB is bits per byte on held-out C4/en (↓); ARC-Easy (ARC-E), HellaSwag (HS), PIQA, and LAMBADA (LM) are accuracies (↑). Bold marks the best result per column among byte-level methods.

		SMALL (85M)					MEDIUM (205M)				
METHOD	ρ	BPB	ARC-E	HS	PIQA	LM	BPB	ARC-E	HS	PIQA	LM
SP-32K		1.068	37.5	34.0	65.6	31.3	1.020	40.8	38.5	68.2	35.3
DATA-MATCHED		1.145	36.7	33.6	64.5	28.4	1.106	39.4	37.2	67.2	32.3
RANDOM	0.225	1.226	34.1	30.9	60.8	21.7	1.171	36.6	34.0	64.4	25.8
SPACE	0.225	1.193	35.4	32.4	62.5	23.3	1.143	37.9	35.2	65.7	28.0
BYTEFLOW	0.225	1.165	36.0	33.1	63.5	26.3	1.121	38.8	36.1	66.5	30.7
H-NET EC	0.225	**1.148**	36.7	**33.7**	**64.6**	**28.4**	1.105	39.4	37.0	67.1	32.4
MLT	0.225	1.149	**36.9**	**33.7**	64.3	28.1	**1.094**	**40.2**	**37.9**	**67.8**	**33.7**

		LARGE (530M)					XL (1.1B)				
METHOD	ρ	BPB	ARC-E	HS	PIQA	LM	BPB	ARC-E	HS	PIQA	LM
SP-32K		0.983	46.5	44.7	70.8	40.6	0.952	47.2	49.2	72.0	43.9
DATA-MATCHED		1.021	44.8	42.7	69.9	37.4	1.004	45.7	48.2	71.0	40.4
RANDOM	0.225	1.133	39.5	36.9	65.2	30.6	1.083	39.6	41.5	67.2	34.1
SPACE	0.225	1.086	42.0	39.7	67.5	32.1	1.047	42.6	44.7	69.7	35.2
BYTEFLOW	0.225	1.064	43.1	40.9	68.3	33.7	1.032	43.3	45.4	70.1	36.8
H-NET EC	0.225	1.050	44.1	42.3	69.3	35.5	1.019	44.4	46.8	70.5	38.6
MLT	0.225	**1.025**	**44.9**	**43.2**	**69.9**	**37.2**	**1.006**	**45.8**	**48.1**	**71.1**	**40.7**

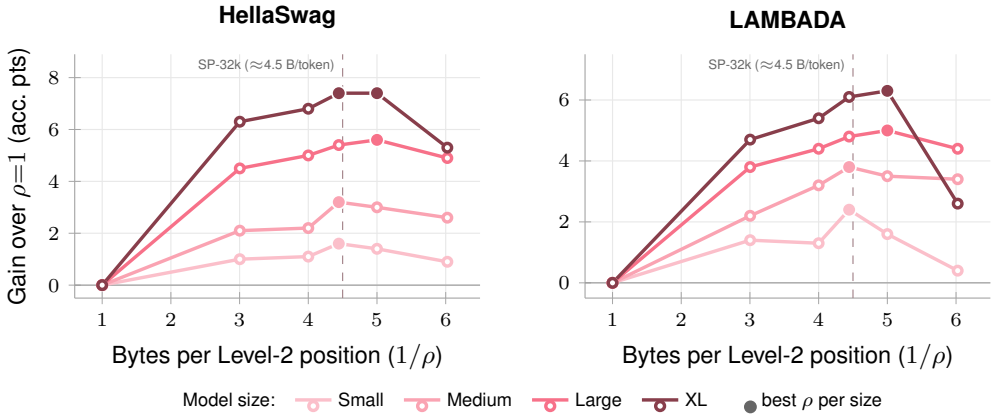

Figure 3: Gain of byte-level MLT over the uncompressed model ($\rho = 1$) as a function of bytes per Level-2 position ($1/\rho$), from Table 1. Darker lines are larger models; filled markers show the best ρ for each size. The dashed line marks the average bytes-per-token of the SP-32k tokenizer (≈4.5).

- **Causal self-attention** ($QK^\top$ and attention-weighted values): $2 \cdot 2T^2 d/2 = 2T^2 d$ FLOPs (the factor of $1/2$ accounts for the causal mask).
- **Attention output projection:** $2d^2 \cdot T = 2Td^2$ FLOPs.
- **SwiGLU MLP** (up, gate, down with $d_{\text{ff}} = \frac{8}{3}d$): $3 \cdot 2d \cdot d_{\text{ff}} \cdot T = 16Td^2$ FLOPs.

Summing,

$$F_{\text{block}}(T) \;=\; 24\,T\,d^2 \;+\; 2\,T^2 d. \tag{10}$$

The output projection (unembedding) adds $2\,T\,d\,V$ FLOPs, where V is the vocabulary size. This cost is non-negligible. For a SP-32k baseline ($V_0 = 32{,}000$) at $d = 768$ (Small), the per-position unembedding cost is comparable to several Transformer blocks.

D.2 Total Forward FLOPs and Matched-Compute Sequence Length

All experiments are FLOP-matched against a dense SP-32k Transformer at sequence length $T_0 = 512$. The total forward FLOPs for this baseline are

$$F_{\text{dense}}(T_0, V_0) \;=\; n \cdot F_{\text{block}}(T_0) \;+\; 2\,T_0\,d\,V_0, \tag{11}$$

where $V_0 = 32{,}000$. For an MLT with vocabulary V (e.g. $V = 256$ for bytes, $V = V_0$ for tokens-as-input), the total is

$$F_{\text{MLT}}(T_1, \rho, V) \;=\; (n_{1,E} + n_{1,D}) \cdot F_{\text{block}}(T_1) \;+\; n_2 \cdot F_{\text{block}}(\rho T_1) \;+\; 2\,T_1\,d\,V. \tag{12}$$

We set $F_{\text{MLT}} = F_{\text{dense}}$ and solve the resulting quadratic for T_1. The exact solution depends on $(\rho, n_{1,E}, n_2, n_{1,D}, d, V, V_0)$ and is computed per-run.

For byte-level models ($V = 256$), the large gap between V and V_0 means the MLT saves substantially on unembedding FLOPs, translating into additional sequence-length headroom beyond what the layer-count ratio alone predicts. This effect is most pronounced at small model sizes, where the unembedding accounts for a larger share of total compute.

To build intuition for the tokens-as-input setting (where $V = V_0$ and the unembedding terms cancel), dropping the quadratic attention term gives a simple approximation:

$$T_1 \;\approx\; T_0 \cdot \frac{n}{(n_{1,E} + n_{1,D}) + \rho \cdot n_2}, \qquad n = n_{1,E} + n_2 + n_{1,D}. \tag{13}$$

This approximation understates the actual headroom for byte-level models, where the vocabulary savings provide significant additional room. We use the exact solution with vocabulary costs in all experiments. For reference, the exact FLOP-matched sequence lengths for our main configurations (with $T_0 = 512$) are:

- Byte-level at $\rho = 0.225$: Medium $T_1 = 1362$ (2.7×), Large $T_1 = 1479$ (2.9×). The large multiplier reflects both the layer-count ratio and the vocabulary savings ($V = 256$ vs. $V_0 = 32{,}000$).
- Token-level at $\rho = 0.8$: Medium $T_1 = 589$ (1.15×), Large $T_1 = 603$ (1.18×). The increase is modest since $V = V_0$ and ρ is close to 1.

D.3 Inference Cost

For autoregressive generation, the per-step forward FLOPs are dominated by the same expressions evaluated at $T =$ current context length. Level 2 contributes only on routed positions; the causal router adds a per-step cost negligible relative to a single Transformer block. KV-cache memory scales with the routed positions in Level 2 (ρT entries per Level-2 layer) and the full sequence in Level 1 (T entries per Level-1 layer). For typical configurations Level 2 dominates the cache, so $\rho < 1$ yields proportional savings. We ignore routing-related FLOPs throughout: the MLP router contributes $< 1\%$ of F_{block} at every scale, and the gather/scatter operation is $\mathcal{O}(Td)$ memory traffic with no FLOPs.

Decoding FLOPs. Ignoring attention, a decoding step costs $24d^2$ FLOPs per executed layer plus $2dV$ for the unembedding. Level 1 runs at every step and Level 2 at a fraction ρ of steps, so relative to a dense Transformer of the same depth and vocabulary

$$\frac{F^{\text{step}}_{\text{MLT}}}{F^{\text{step}}_{\text{dense}}} \;=\; \frac{24d^2\,(n_{1,E} + n_{1,D} + \rho\, n_2) + 2dV}{24d^2\,n + 2dV}. \tag{14}$$

For the token-level Large model ($d = 1408$, $n = 22$, $n_2 = 18$, $V = 32{,}000$) this ratio is 0.85 at $\rho = 0.8$ and 0.77 at $\rho = 0.7$. A Level-2 layer attends only at routed steps and then only over the $\approx \rho t$ cached routed positions, so its expected attention FLOPs scale as ρ^2 and its KV cache as ρ. At long

contexts, where attention dominates, the relative cost therefore approaches $(n_{1,E} + n_{1,D} + \rho^2 n_2)/n$, i.e. 0.71 at $\rho = 0.8$ and 0.58 at $\rho = 0.7$, and the relative KV-cache size is $(n_{1,E} + n_{1,D} + \rho\, n_2)/n$, i.e. 0.84 and 0.75. These are analytical estimates; realized speedups also depend on gather/scatter overheads and kernel efficiency, which we have not benchmarked. For byte-level models the relevant comparison is per unit of text rather than per step: at $\rho = 0.225$ a Large byte-level MLT spends $\approx 1.5\times$ the non-attention FLOPs of the SP-32k model per $\sim$4.5 bytes, mirroring the training-time FLOP match in which it processes $T_1 = 1479$ bytes for the budget the SP-32k model spends on 512 tokens.

E H-NET ARCHITECTURE ABLATION

We refer to our controlled re-implementation of H-Net (Hwang et al., 2025) as H-Net EC, where EC stands for Expert-Choice routing. It shares MLT's network skeleton and training recipe but replaces MLT's chunking mechanisms with H-Net's. The goal is to isolate the contribution of the chunking mechanisms from the shared architecture: both models use the same Level-1 encoder / Level-2 / Level-1 decoder split, the same layer counts (Table 5), expert-choice top-k routing, and identical training details (optimizer, schedule, batch size, FLOP matching). The only differences are listed below.

How H-Net EC differs from MLT.

- **Cosine-similarity routing policy.** Instead of the learned MLP scoring head, the router score is $p_t = \frac{1}{2}(1 - \cos(W_k h^E_{t-1}, W_q h^E_t))$, with learned projections W_k, W_q.
- **EMA smoothing.** Selected Level-2 inputs are smoothed by an exponential moving average across adjacent selected positions: $\bar{z}_i = P_i \hat{z}_i + (1 - P_i)\bar{z}_{i-1}$, where i indexes the selected positions.
- **Broadcast de-chunking.** Instead of MLT's scatter, compressed vectors are broadcast back to byte resolution by causal expansion ($\tilde{z}_t = \bar{z}_{\sum_{k \le t} b_k}$), scaled by a per-position confidence c_t and a straight-through estimator at the merge.
- **Zero-initialized residual projection.** A learned linear projection is applied to the encoder residual before it is added to the dechunked Level-2 output (Hwang et al., 2025, Eq. 3). This projection is initialized close to zero, so that at initialization the decoder receives primarily the Level-2 output and the encoder residual contributes nothing.
- **Inter-stage RMSNorm.** An RMSNorm layer is applied at the output of every encoder, decoder, and Level-2 component.

Note that the original H-Net enforces capacity through a ratio loss $\mathcal{L}_{\text{ratio}}$ that pushes the realized compression toward a target $1/N$. We replace this with expert-choice top-k (shared with MLT), which enforces capacity by construction and removes the ratio loss. Any difference between MLT and H-Net EC in Section 4.2 is therefore attributable to the chunking mechanisms above.

F BYTEFLOW BASELINE

ByteFlow (Deng et al., 2026) promotes positions to its upper level according to their marginal lossy coding rate. For Level-1 encoder states $H = h^E_{1:T} \in \mathbb{R}^{T \times d}$, the coding rate of a prefix is $R(h^E_{1:t}) = \frac{1}{2} \log\det\big(I_t + \frac{d}{\epsilon^2} h^E_{1:t} (h^E_{1:t})^\top\big)$, and the score of position t is its increment $s_t = R(h^E_{1:t}) - R(h^E_{1:t-1})$. All increments follow from a single Cholesky factorization $LL^\top = I_T + \frac{d}{\epsilon^2} HH^\top$ as $s_t = \log L_{tt}$. To compare only the chunking criterion, ByteFlow (adapted) keeps everything else from MLT: the same skeleton and layer counts, expert-choice top-k on s_t with $k = \rho T$, scatter-based merging, and identical training details. Because the score has no parameters and receives no gradient, the merge is unweighted, as for Random and Space, and the score is computed on the stop-gradient encoder output. At inference we use the same causal router as MLT, trained to predict the top-k selection. The factorization costs $\mathcal{O}(T^3)$ FLOPs per sequence, well below 1% of the model FLOPs at our sequence lengths, and like the router it is excluded from FLOP matching.

G Architectural Ablations

We perform a leave-one-out ablation to isolate the contribution of each design choice that distinguishes MLT from H-Net EC. Starting from MLT, we replace one component at a time with its H-Net counterpart and measure the effect on downstream performance. The bundled comparison is in Section 4.2; the figure below decomposes that result by mechanism.

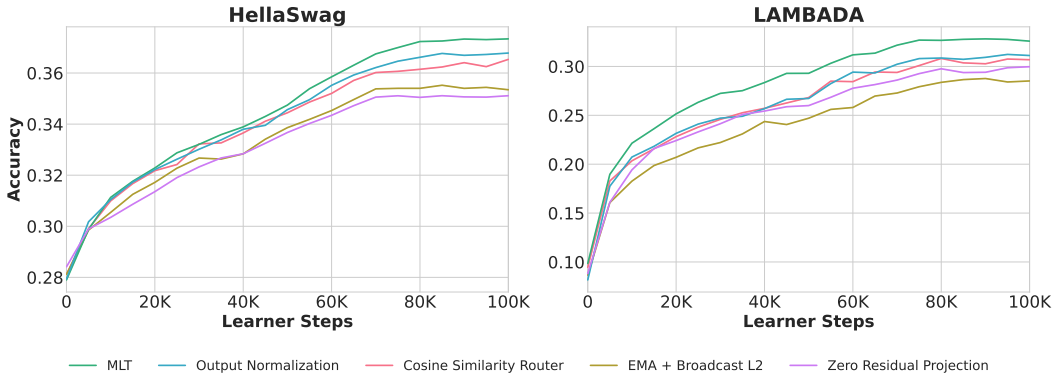

Figure 4: Leave-one-out ablation on the Medium model at $\rho = 0.225$. Starting from MLT, each curve replaces one design choice with its H-Net counterpart and shows the resulting training trajectory on HellaSwag (left) and LAMBADA (right).

The ablated components are:

- **Router input: h_t^E vs. (h_{t-1}^E, h_t^E).** Concatenating adjacent encoder embeddings as router input provides an inductive bias toward state-change boundaries, similar to H-Net's cosine-similarity policy. This improves performance at small scale but the benefit tapers at larger model sizes; we opt for the simpler single-input formulation.
- **Router policy: MLP vs. cosine similarity.** Replacing the learned MLP scoring head with a cosine-similarity policy.
- **Gather/scatter vs. broadcast.** Replacing the scatter-merge (Eq. 2) with a broadcast-and-confidence upsampler.
- **EMA smoothing.** Adding an EMA over adjacent compressed vectors.
- **Inter-stage RMSNorm.** Adding RMSNorm at the output of the Level 1 Encoder and Level 2.
- **Level-1 layer count.** Sweeping total Level-1 layers $n_1 \in \{2, 4, 6, 8\}$ (with $n_{1,E} = n_{1,D} = n_1/2$) at fixed total depth, compensating in n_2. See Appendix H for the full analysis including inference-cost implications.
- **Value weighting by σ_t.** Multiplying values by the routing score σ_t in every Level-2 attention layer. See Appendix I for a dedicated comparison.

H Level-1/Level-2 Layer Split

We sweep the total number of Level-1 layers $n_1 = n_{1,E} + n_{1,D}$ (with $n_{1,E} = n_{1,D}$) from 2 to 8 at fixed total depth, compensating in n_2. All configurations are FLOP-matched at training time by adjusting T_1 (Appendix B.3).

Table 7: Layer split sweep (FLOP-matched at training). n_1 is the total number of Level-1 layers.

		Bytes ($\rho = 0.225$)				Tokens ($\rho = 0.8$)			
		Medium		Large		Medium		Large	
n_1	n_2	HS	LM	HS	LM	HS	LM	HS	LM
2	$n-2$	37.1	33.2	42.4	**37.5**	38.1	34.0	44.6	38.5
4	$n-4$	**37.9**	**33.7**	**43.2**	37.2	38.8	35.6	45.1	40.8
6	$n-6$	36.6	32.7	41.2	36.3	39.1	36.1	45.6	**41.8**
8	$n-8$	36.1	30.5	41.1	36.3	**39.3**	**36.9**	**45.7**	41.7

The Medium model has $n = 16$ total layers and the Large model has $n = 22$. The two input settings show different trends. For byte-level modeling, performance peaks at $n_1 = 4$ in three of the four columns and degrades with more Level-1 layers, particularly at Medium scale. For token-level modeling, performance generally increases with n_1, suggesting that already-compressed inputs benefit from stronger preprocessing in Level 1.

While all configurations are FLOP-matched at training (by adjusting the Level-1 sequence length T_1, see Appendix B.3), the layer split also affects inference cost. In practice, models are deployed at context lengths much longer than the training sequence length, so the total compute during inference depends on the actual layer counts rather than on the FLOP-matched training budget. Increasing the number of Level-1 layers means every position in the sequence passes through more layers, including the non-routed majority. This disproportionately increases the cost of the "cheap" path, reducing the overall compute savings that compression provides. The token-level gains from deeper Level 1 must therefore be weighed against reduced inference savings.

We keep $n_1 = 4$ ($n_{1,E} = n_{1,D} = 2$) throughout our main experiments as a good trade-off between training performance and inference efficiency. We also find that splitting the budget asymmetrically (e.g. $n_{1,E} = 1, n_{1,D} = 3$) does not meaningfully affect performance; only the total n_1 matters. Concurrent work on whitespace-segmented byte hierarchies similarly varies where the byte-level layers sit (before, after, or interleaved with the word-level layers) and finds only modest differences on general benchmarks (Edman & Fraser, 2026).

I Value Weighting by Router Scores

Following Lou et al. (2024), we multiply the value vectors in every Level-2 attention layer by the corresponding router score σ_t. This is not critical to the algorithm but provides a small, consistent improvement across model sizes and stabilizes early training. The effect can be understood through the dual role of the routing score: on the query side it determines which positions invoke Level 2 (adaptive computation), and on the key/value side it modulates the contribution of each position to the attention context (compression). Weighting the values by σ_t makes this second role explicit in every attention layer rather than only at the merge step.

Figure 5 shows the effect of removing value weighting and the backward loss ($\lambda = 0.05$) separately. Removing the backward loss has a more noticeable impact on sampling-based evaluations (LAMBADA), while removing value weighting leads to a small reduction on both scoring (HellaSwag) and sampling benchmarks. These results are consistent across model sizes and datasets.

J Annotated Chunk Boundaries

We show two C4/en excerpts with MLT routing decisions at $\rho = 0.225$. Each byte is shown in monospace: `green` for positions selected by the router (routed to Level 2), `red` for positions that retain the encoder residual only. The model receives no tokenizer supervision; all boundaries emerge from the language-modeling objective alone.

Example 1: Informal blog text. `After spending quite sometime at the hotel and surrounding area, we then headed to Belle Isle. We were able to shoot there not to long ago with another photographer couple which I need to blog because their pictures were such a favorit`

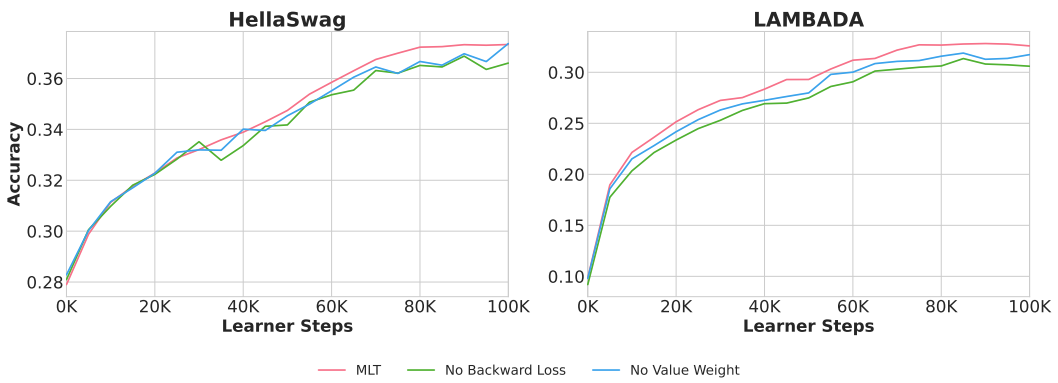

Figure 5: Ablation of value weighting and backward loss on the Medium model at $\rho = 0.225$. Removing the backward loss ($\lambda = 0.05$) primarily affects sampling-based evaluation (LAMBADA); removing value weighting reduces performance on both benchmarks by a small margin.

`e! Belle Isle is also amazing. The bridge that you walk is so fun and even with people walking in and out of your shots, it is perfect.`

Most boundaries coincide with word boundaries (spaces); within longer words the model discovers morpheme-like splits: `surrounding`, `headed`, `hotographer`, `ictures`, `favorite`. Punctuation and sentence boundaries are consistently selected. The average chunk length is ~ 4.5 bytes, matching the bytes-per-token of SentencePiece-32k on this text.

Example 2: Legal/formal text. `will not consent to any Advertising which, in its absolute discretion, contains inappropriate artwork from the Event or Youth Provision Organiser. An Event or Youth Provision Organiser shall not be entitled to a refund of any Fees incurred in relation to the Advertising and/or Email Distribution where that Event or Youth Provision Organiser fails to meet the Y-DAP/www.ydap.org.uk requirements in relations to Advertising by the deadline for submission of the same.`

Three features stand out. **Consistency:** the repeated phrase `Youth Provision Organiser` is chunked identically across all four occurrences; `Advertising` is split the same way in all three. **Morphological awareness:** Latin suffixes are isolated (`ion`, `ing`, `d`, `s`), and the model correctly treats "and/or" as a single chunk. **Structured substrings:** the URL `www.ydap.org.uk` is chunked at every dot, mirroring the hierarchical structure of domain names.

Example 3: Token-level routing. When MLT operates on tokens as inputs, the router decides which tokens pass through Level 2 and which are handled by Level 1 alone. Below we show two excerpts at different compression rates.

At $\rho = 0.6$ (skipping $\sim$40% of tokens):

`Ben & I (and the kids) are thrilled with our decision and we just can't wait to move in. So thank you for your professionalism & most of all, your patience. We appreciate all you did for us. And finally thank you for making our first purchase in the USA a good experience. We certainly will have no hesitation in recommending you to our friends`

At $\rho = 0.7$ (skipping $\sim$30% of tokens):

Conditions were ideal for drivers to fine-tune their set-ups and understand how their tyres handled the roughest track in the program. With only six weeks to the start of the championship, performance has also become a priority. Nyck de Vries (Koiranen GP) took the lead without delay. The young Dutch driver managed to keep Jake Dennis (Fortec Motorsports), Oliver Rowland (Manor MP Motorsport) and Bruno Bonifacio (Prema Powerteam) at bay.

In both cases, the router consistently skips function words and punctuation (the, of, in, to, for, and, a, ., ,) while routing content-carrying tokens to Level 2. At higher compression ($\rho = 0.6$), more words are skipped but the core meaning is preserved. The selected tokens read like compressed, telegraphic text that retains the informative content while stripping predictable scaffolding.

K Token Count Stability

A fixed per-item compression rate ρ assumes that information density is roughly uniform across sequences of a given length. To verify this, we sample 10,000 byte sequences at each length uniformly from C4/en and measure the distribution of SP-32k bytes-per-token (Figure 6). As the byte sequence length increases, the distribution tightens and converges to $\sim$4.5 bytes per token, matching the optimal compression rate discovered by MLT. This confirms that for sufficiently long sequences, information density is stable enough that a fixed ρ per item is a reasonable design choice.

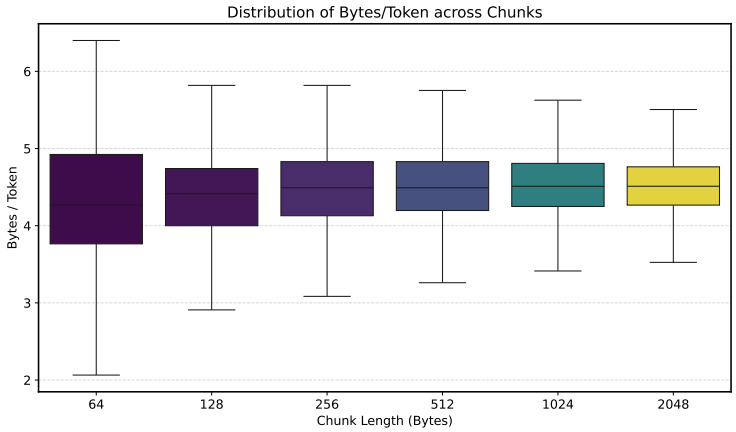

Figure 6: Distribution of bytes per token across byte sequences of different lengths on C4/en with SP-32k. As sequence length increases, the distribution tightens and converges to $\sim$4.5: information density smooths out at the scales we operate at.

L Neuroscience Inspiration

A key motivation for MLT comes from how the brain processes temporal experience. Research on event segmentation (Zacks & Tversky, 2001; Radvansky & Zacks, 2017) shows that the brain actively parses continuous experience into discrete, hierarchical events. Lower-level sensory regions detect fine-grained events; these compose upward through the cortical hierarchy, where signals from different modalities are fused and events become progressively more abstract (Baldassano et al., 2017; Ben-Yakov & Dudai, 2011). At the highest levels, event boundaries serve as access points for episodic memory (Michelmann et al., 2023). The boundaries themselves are shaped by the tasks being solved, not by any single sensory cue (Shim et al., 2024).

MLT's architecture is loosely inspired by this hierarchical structure. Level 1 processes every input position, similar to how low-level sensory regions see everything. The router identifies salient positions, and Level 2 operates only at those positions, integrating information at a coarser temporal resolution. We stress that this is an architectural analogy. We do not model biological event segmentation or hippocampal memory; we simply adopt the principle that hierarchical, task-driven compression is a useful way to organize computation in sequence models.