DYNAMIC LARGE CONCEPT MODELS: LATENT REASONING IN AN ADAPTIVE SEMANTIC SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models apply their full computational stack to every token, although the information carried by a sequence varies across spans. We introduce Dynamic Large Concept Models (DLCM), which form variable-length concepts from contextual token representations and process them with a larger concept-level backbone. A lightweight causal decoder uses completed concepts to generate subsequent tokens autoregressively, sharing concept-level computation across multiple predictions. Global compression regularization allows segment lengths to adapt to content while regulating their average. We also introduce a compression-aware scaling law that separates token capacity, concept capacity, and data, together with a decoupled maximal-update parametrization for modules of different widths. A 2.3B-parameter DLCM improves the mean accuracy over a 1.3B dense baseline by 1.01 percentage points across twelve zero-shot benchmarks, and its estimated forward FLOPs are 26.0% lower at a 32K context (11.8% at the 8K training context), with savings that grow with sequence length. Both models are trained on one trillion tokens. Gains are strongest on commonsense and question-answering tasks, while compression and token-loss analyses reveal sensitivity to segment granularity. The scaling analysis further predicts lower loss at equal compute through concept-level capacity allocation.

1 INTRODUCTION

Language carries information unevenly. Locally predictable spans alternate with transitions that introduce a new idea, relation, or reasoning step. Yet standard autoregressive Transformers apply their full computational stack to every token (Brown et al., 2020; Touvron et al., 2023). Dynamic Large Concept Models (DLCM) separate the granularity of deep reasoning from that of token generation: lightweight token modules preserve local detail, while a larger backbone processes a shorter sequence of variable-length concepts (Figure 1).

Intermediate granularity determines where computation is spent. Fixed chunks impose uniform spans, while sentence-level models use predefined linguistic units (LCM team et al., 2024). H-Net learns content-dependent compression in an autoregressive hierarchy (Hwang et al., 2025). DLCM builds on this direction in a token-based model that combines dynamic concept formation, concept-level causal reasoning, and token-level decoding under a next-token objective.

We use *concept* to mean a contiguous token segment represented by one latent vector, without requiring a linguistic boundary. A cosine rule identifies contextual changes; the *Global Parser* regulates their aggregate rate while allowing content-dependent segment lengths. Completed concepts support several subsequent token predictions through causal cross-attention, sharing deep computation while retaining autoregressive generation.

Compression also changes how capacity should be allocated. Parameters in a concept backbone operate over fewer sequence positions than parameters in a token module, but greater compression makes the representation bottleneck more restrictive. Our compression-aware scaling law models token capacity, concept capacity, compression, and training data separately. A decoupled maximal-update parametrization (μP) supports training at unequal module widths and transferring learning rates from a small proxy to larger models (Yang et al., 2022).

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.

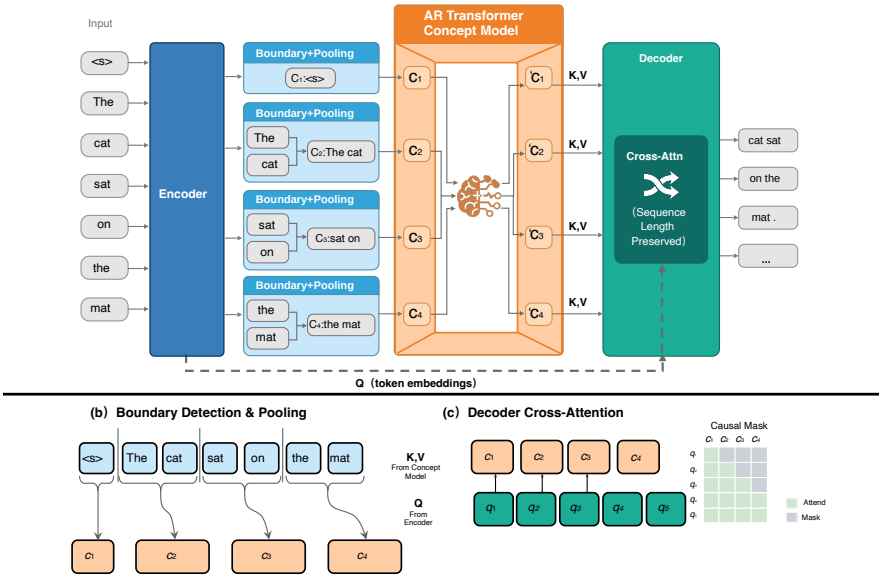

Figure 1: **Dynamic concepts connect token-level encoding and decoding.** The encoder and pooling stages form concepts; the concept Transformer supplies states for subsequent token predictions. Bottom: boundary pooling and causal cross-attention, with token queries and completed-concept keys/values. Each output phrase shares concept computation while its tokens are generated autoregressively. A concept is available only after the boundary that closes it is observed (Equation 5).

A 2.3B DLCM and a 1.3B dense baseline are trained on the same one trillion tokens. DLCM improves eight of twelve zero-shot scores and raises mean accuracy by 1.01 percentage points. Its estimated forward FLOPs are 26.0% lower than the baseline's at a 32K context and 11.8% lower at the 8K training context (Figure 9). Its larger concept backbone converts sequence compression into additional reasoning capacity. The loss–FLOPs comparison further predicts lower loss at matched compute budgets (Appendix D.4), connecting concept-level capacity allocation to scaling.

2 Related Work

Hierarchical and concept-level language models. Large Concept Models predict sentence representations in a pretrained embedding space and decode them back into language (LCM team et al., 2024). DLCM instead learns its token representations jointly with the language model, forms variable-length segments, and optimizes token likelihood. H-Net is the closest architectural precedent: it combines dynamic chunking, a compressed main network, and autoregressive decoding (Hwang et al., 2025). We adopt its adjacent-representation boundary principle and ratio-regularization formulation. DLCM's focus is their integration into a heterogeneous token/concept Transformer with global compression statistics, concept-to-token cross-attention, and an explicit study of capacity allocation and scaling. Byte Latent Transformer also allocates computation through variable-length patches, using an entropy-based patching mechanism over bytes (Pagnoni et al., 2025). These methods share the objective of concentrating computation at a resolution adapted to the input; their input units, boundary signals, and decoding mechanisms differ.

Latent reasoning and adaptive computation. Continuous latent reasoning methods can feed hidden states through successive reasoning steps before emitting an answer (Hao et al., 2025). DLCM's concept backbone processes a compressed representation of the observed sequence inside next-token prediction. We use *concept-level reasoning* for this deeper computation over latent segments, without assuming that each segment represents an independently decoded reasoning step. Adaptive-depth architectures and mixture-of-experts models vary computation along other axes, through iterative refinement or parameter routing (Dehghani et al., 2018; Shazeer et al., 2017). DLCM varies sequence resolution. Its scaling formulation extends the capacity–data decomposition of language-

model scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), while its optimization follows the feature-learning perspective of μP (Yang et al., 2022).

3 Dynamic Large Concept Models

3.1 Token representations and dynamic boundaries

For tokens $x_{1:L}$, DLCM models $p(x_{1:L}) = \prod_t p(x_t \mid x_{<t})$. Its causal encoder produces $\mathbf{h}_t = \mathcal{E}(x_{\le t}) \in \mathbb{R}^{d_{\text{tok}}}$ for segmentation and decoding. The concept backbone can use a larger width d_{con} because it processes fewer positions.

Following the adjacent-representation principle of H-Net (Hwang et al., 2025), we compute a boundary score from projected encoder states:

$$\mathbf{q}_t = W_q \mathbf{h}_t, \qquad \mathbf{k}_t = W_k \mathbf{h}_t, \qquad p_t = \frac{1 - \cos(\mathbf{q}_{t-1}, \mathbf{k}_t)}{2}, \qquad p_1 = 1. \tag{1}$$

Here $W_q, W_k \in \mathbb{R}^{d_{\text{scan}} \times d_{\text{tok}}}$, and $p_t \in [0, 1]$ measures a change between adjacent contextual representations. It uses only the current token and its prefix, preserving causal encoding. No look-ahead is required: b_t is decided once x_t is observed, and a concept is exposed to the decoder only after the boundary that closes it has been observed (Equation 5).

Let $b_t = 1$ mark a segment start. Training samples $b_t \sim \text{Bernoulli}\big(\min(1, p_t/\alpha)\big)$ with sharpening temperature $\alpha = 0.95$; inference uses $b_t = \mathbb{I}[p_t \ge 0.5]$. No gradient flows through the sampled indicators b_t. The ratio loss $\mathcal{L}_{\text{aux}}$ (Section 3.2) acts on the continuous scores p_t and controls the boundary rate, while where boundaries fall is determined by the encoder states $\mathbf{h}_t$, which are trained by next-token prediction. A separate learned predictor motivates this choice in Section 5.5.

If the resulting segments are $S_1, \ldots, S_M$, each concept pools one segment and projects it into the backbone width:

$$\mathbf{c}_j = W_{\text{up}} \left(\frac{1}{|S_j|} \sum_{t \in S_j} \mathbf{h}_t \right), \qquad W_{\text{up}} \in \mathbb{R}^{d_{\text{con}} \times d_{\text{tok}}}. \tag{2}$$

The realized compression ratio is $\widehat{R} = L/M$. A target $R = 4$ requests four tokens per concept on average; it does not prescribe a four-token length for every segment or imply $\widehat{R} = 4$ on every example.

3.2 Global compression control

The Global Parser aggregates boundary statistics across the distributed batch and accumulation window. Load-balancing feedback lowers scores when the boundary rate exceeds $1/R$ and raises them when it falls below, allowing individual sequences to use different segment lengths. Training combines next-token prediction and compression control:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{aux}}, \tag{3}$$

where λ weights the ratio loss of H-Net (Hwang et al., 2025), applied here to global statistics. Appendix A gives its form and feedback direction. The encoder, pooling projection, backbone, and decoder learn through their continuous computations; realized compression remains content-dependent.

3.3 Concept processing and autoregressive decoding

The causal concept Transformer computes $\mathbf{Z} = \mathcal{M}(\mathbf{C})$ over M pooled representations, updating when a new concept becomes available. Completed concepts support subsequent autoregressive token predictions. After causal smoothing $\widetilde{\mathbf{Z}} = \mathcal{S}(\mathbf{Z})$, where $\mathcal{S}$ is a Mamba layer (Gu & Dao, 2023) applied causally over the concept sequence, token states supply queries and concept states supply keys and values:

$$Q = HW_Q, \qquad K = \widetilde{Z}W_K, \qquad V = \widetilde{Z}W_V, \qquad U = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}} + A \right) VW_O + H. \tag{4}$$

For one head, $W_Q \in \mathbb{R}^{d_{\text{tok}} \times d_h}$, $W_K, W_V \in \mathbb{R}^{d_{\text{con}} \times d_h}$, and $W_O \in \mathbb{R}^{d_h \times d_{\text{tok}}}$. Token-level causal decoding maps U to the next-token distribution. RMS normalization of attention queries and keys stabilizes the interface between the two widths (Henry et al., 2020; Dehghani et al., 2023).

Causality applies to concept formation as well as attention. Define a_j as the earliest token position at which concept j can be constructed from the observed prefix, including the boundary decision that makes it available. For the query at position t predicting x_{t+1}, the mask must satisfy

$$A_{t,j} = \begin{cases} 0, & a_j \le t, \\ -\infty, & a_j > t. \end{cases} \tag{5}$$

The mask distinguishes segment membership from concept availability: a pooled concept cannot predict the tokens used to form it. If the next boundary closes a segment, availability includes that observation. The same ordering governs packed training and cached generation. Training and generation use the same replicated keys and values (Section 3.4): during generation, each new token position appends a copy of the most recent available concept to the cross-attention cache, and the concept backbone runs only when a boundary closes a new concept. Attention weights, including replica multiplicities, therefore match between training and generation.

For example, concept c_1 supports several subsequent tokens, each also conditioned on the generated prefix. When c_2 becomes available, it updates the concept context. Figure 1 illustrates this sharing; Appendix B formalizes the alignment.

3.4 Efficient attention across resolutions

Using `repeat_interleave`, replication expands concept keys and values into runs aligned with concept availability: concept j occupies key positions from a_j onward, so a standard causal FlashAttention kernel (Dao et al., 2022) enforces Equation 5 without a custom mask (Figure 10). Softmax weighting depends on the visible copies of each concept (Appendix B.1). The microbenchmark compares this implementation with dynamic-mask FlexAttention, measuring kernel speed separately from model-level FLOPs.

4 Optimization and Compression-Aware Scaling

4.1 Decoupled maximal-update parametrization

Different token and concept widths introduce two scaling factors. Relative to the reference width d_{base}, define $s_{\text{tok}} = d_{\text{tok}}/d_{\text{base}}$ and $s_{\text{con}} = d_{\text{con}}/d_{\text{base}}$. We adapt μP (Yang et al., 2022) by applying width-dependent learning rates separately to the token modules and concept backbone:

$$\eta_{\mathcal{E},\mathcal{D}} = \frac{\eta_{\text{tok}}^{\text{base}}}{s_{\text{tok}}}, \qquad \eta_{\mathcal{M}} = \frac{\eta_{\text{con}}^{\text{base}}}{s_{\text{con}}}. \tag{6}$$

Within each width group, hidden-weight initialization variance and AdamW's ϵ follow inverse-width scaling. Embeddings and biases retain their base learning rates, and the output logits are scaled by s_{tok}^{-1}. Appendix C collects the parametrization and tuning protocol.

Coordinate sweeps tune base rates on an 87M proxy, then transfer them to 274M, 468M, and approximately 834M models. Perturbations away from this setting increase loss (Figure 2). Similar preferred base rates yield different actual rates through Equation 6, supporting decoupled scaling across the evaluated widths.

4.2 Separating capacity, compression, and data

Compression creates two capacity bottlenecks: processing and reconstructing tokens, and modeling the relationships between their compressed representations. Let N be the total parameter count and P the share assigned to the concept backbone. Thus $N_{\text{tok}} = N(1-P)$ and $N_{\text{con}} = NP$ describe the complementary token/interface and concept capacities. Let D denote the number of training tokens represented by a loss observation. We extend the capacity–data structure of Chinchilla (Hoffmann et al., 2022) with separate capacity terms and an explicit compression dependence:

$$\widehat{\mathcal{L}}(N, D, R, P) = E_0 + \frac{A_{\text{tok}}}{(N(1-P) + t_{\text{tok}})^{\delta_{\text{tok}}}} + \frac{A_{\text{con}} R^{\gamma}}{(NP + t_{\text{con}})^{\delta_{\text{con}}}} + \frac{A_{\text{data}}}{(D + t_{\text{data}})^{\alpha}}. \tag{7}$$

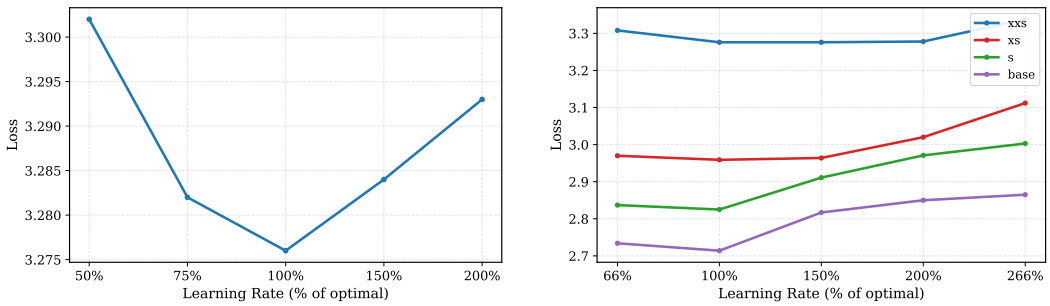

Figure 2: **Learning-rate transfer with decoupled μP.** Left: concept-rate sweep on the 87M proxy. Right: joint token/concept-rate rescaling across model sizes. The common minimum supports transfer across widths; Appendix C gives the size mapping.

Here E_0 is a loss floor, the A coefficients set each contribution's amplitude, and the δ exponents describe capacity dependence. The data exponent is α; the t terms are finite-size offsets in parameter or token units. The exponent γ captures dependence on segment granularity at fixed capacity.

At fixed N and P, changing R changes representation granularity; under fixed compute, it also permits reallocating capacity. A cost model connects these effects. At $R = 1$, the explicit compression factor vanishes, but two capacity terms remain. Appendix D gives the assumptions that reduce them to a single Chinchilla term.

All scaling runs use a warmup–stable–decay (WSD) learning-rate schedule (Hu et al., 2024). During the stable phase, the loss reached after D tokens approximates that of a constant-rate run of length D, so one trajectory supplies many data points (Hägele et al., 2024); the final decay produces an additional loss drop, which we model separately (Tissue et al., 2024; Luo et al., 2025). Equation 7 is therefore fitted to stable-phase observations, and the decay correction of Appendix D.2 maps them to final losses. Because D varies along a trajectory rather than across independently tuned runs, α is a within-trajectory data exponent and is not directly comparable to the Chinchilla exponent.

The plotted trajectories comprise nine runs: three model sizes crossed with $R \in \{2, 4, 8\}$. Each reaches 167.77B tokens; the backbone shares are 31.68%, 33.04%, and 41.70% at the three sizes. Re-evaluating the saved coefficients on the stable-phase observations gives pooled $R^2 = 0.9875$ and RMSE 0.0766. The late-decay model predicts final smoothed loss with RMSE 0.0028 across these nine runs. Figures 3 and 4 show the trajectories and decay behavior. Appendix D reports coefficients, units, preprocessing, and per-run checks. These are within-sample reconstructions; size and allocation vary together in this plotted subset.

4.3 Compute-optimal allocation

The relevant comparison is the cost of processing the same token stream. A sequence-level accounting separates

$$\mathcal{F}_{\mathrm{DLCM}} = \mathcal{F}_{\mathcal{E}} + \mathcal{F}_{\mathrm{segment}} + \mathcal{F}_{\mathcal{M}} + \mathcal{F}_{\mathcal{D}} + \mathcal{F}_{\mathrm{head}}. \tag{8}$$

The concept backbone processes $M = L/\widehat{R}$ positions. At fixed width and depth, its projections and feed-forward work scale approximately as $1/\widehat{R}$, and full-sequence attention pairs as $1/\widehat{R}^2$. Cached generation instead updates the backbone on concept arrivals; the quadratic factor is not a per-token decoding speedup. Compression pays for a wider backbone while token modules, interfaces, and the output head retain their own costs.

The analytical sweeps use the parameter-linear training proxy $\mathcal{C}_{\mathrm{lin}} = 6D_0N[(1-P) + P/R]$, with raw parameter and token counts. At fixed D_0 and compute budget B, the allocation objective is

$$\widehat{\mathcal{L}}^{\star}(B; D_0) = \min_{N,P,R:\, \mathcal{C}(N,D_0,R,P) \le B} \widehat{\mathcal{L}}(N, D_0, R, P), \tag{9}$$

where $\mathcal{C}$ specifies the cost convention. Figure 8 (Appendix D.4) evaluates the analytical-projection coefficients under $\mathcal{C}_{\mathrm{lin}}$; Table 10 lists matched-budget values from the same curves. The fixed $P = 37.4\%$, $R = 2$ setting follows a separate equal-loss grid search.

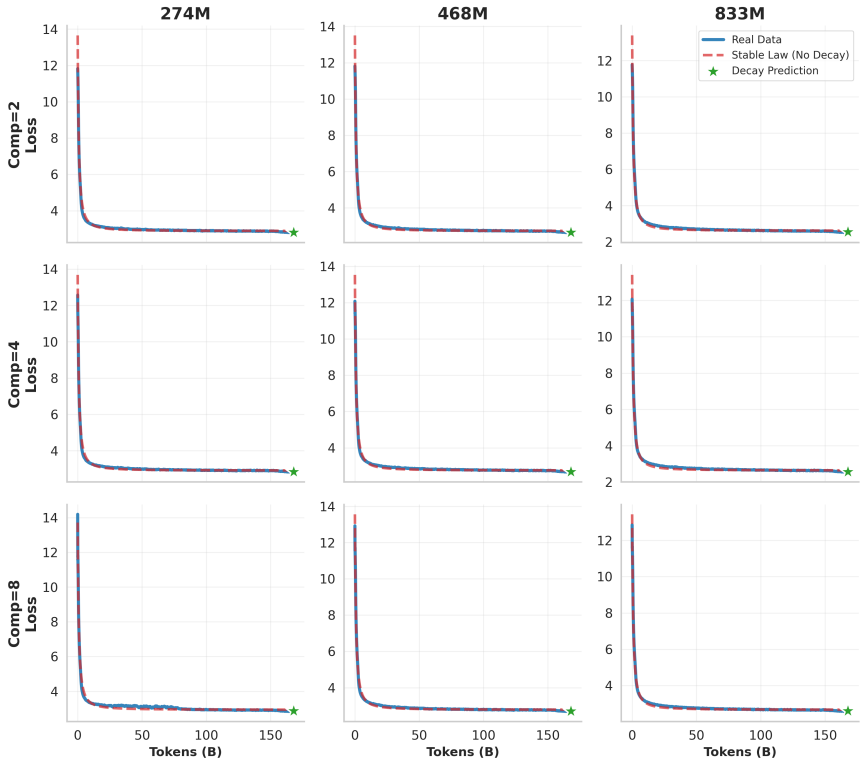

Figure 3: **Compression-aware scaling across sizes and granularities.** Columns: 274M, 468M, 833M; rows: $R = 2, 4, 8$. Blue curves are 100-step mean training losses, dashed curves stable-phase predictions, and stars simplified decay predictions. Stable-phase pooled $R^2 = 0.9875$; Appendix D.2 defines the evaluation window.

At $R = 4$, a complete module-level count (Table 11) under exact causal attention pairs with replicated decoder keys and values gives the 2.3B DLCM 11.8% fewer forward FLOPs than the 1.3B dense baseline at the 8K training context, 18.5% at 16K, and 26.0% at 32K. The concept backbone is twice as wide but processes a quarter of the positions, so its projection and feed-forward cost roughly matches that of token-width layers; the saving comes from attention products, which scale with M^2. The saving therefore grows with context length (Figure 9). In cached generation, the backbone runs only when a boundary closes a concept, and the tokens that follow reuse it (Figure 5). These are FLOPs counts; Appendix H reports attention-kernel timings separately.

Appendix D.3 presents the separate analytical allocation and FLOPs sweeps, with their comparison settings stated explicitly.

5 Experiments

5.1 Pretraining and Comparison Protocol

Both main models train from scratch on one trillion tokens using the DeepSeek-V3 tokenizer (Liu et al., 2024). Their shared mixture combines English and Chinese web text, code, and mathematics from Nemotron-CC, MAP-CC, OpenCoder, and MegaMath (Su et al., 2025; Du et al., 2024; Huang et al., 2025; Zhou et al., 2025). Appendix E lists the corpus proportions and the configurations for each analysis.

The baseline follows the LLaMA architecture (Touvron et al., 2023); DLCM uses the Global Parser with $R = 4$. Both have 32 Transformer layers, but DLCM allocates them between narrower token modules and a wider concept backbone (Table 1). Training tokens are held fixed; total parameters differ.

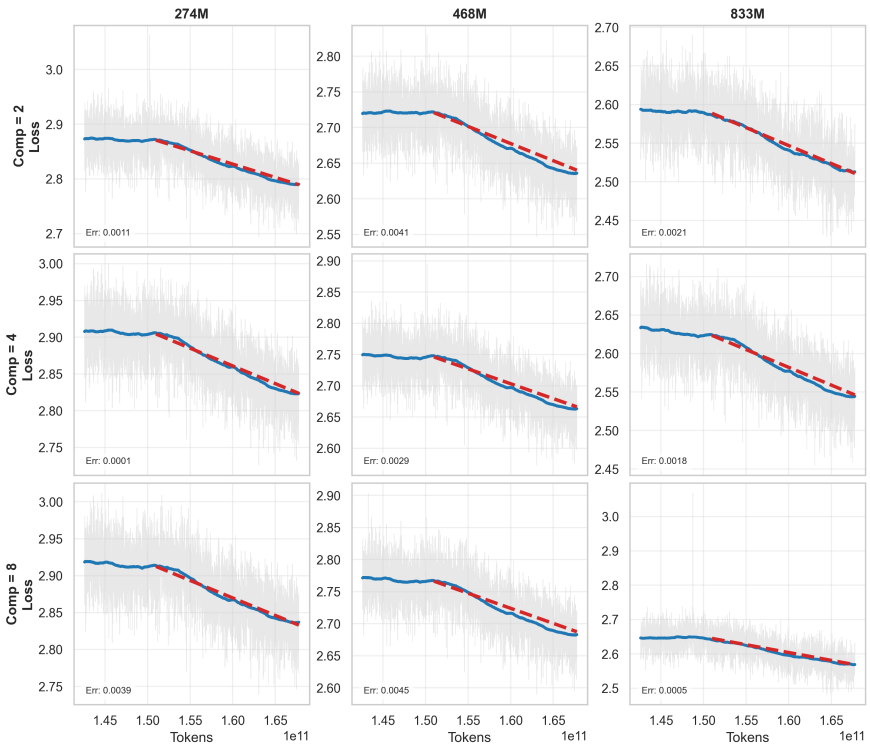

Figure 4: **Scaling through the final learning-rate decay.** Blue curves show geometrically smoothed training losses; red dashed lines connect the pre-decay mean to the robust model's predicted endpoint. Across nine runs, endpoint RMSE is 0.0028. The robust model differs from the star predictions in Figure 3 (Appendix D.2).

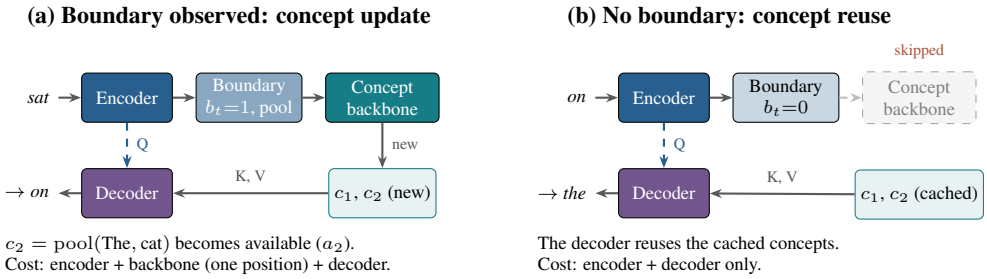

Figure 5: **Cached generation of "The cat sat on the".** (a) When the boundary after "cat" is observed at "sat", the concept backbone runs once to add $c_2 = \mathrm{pool}(\mathrm{The}, \mathrm{cat})$, and the decoder predicts "on". (b) At "on" no boundary is observed; the backbone is skipped and the decoder attends to the same cached concepts to predict "the". A concept is never used to predict the tokens it pools (Equation 5).

5.2 Zero-shot benchmark results

Table 2 shows gains on eight of twelve tasks. Mean accuracy rises from 41.23% to 42.24%, a gain of 1.01 percentage points. The largest gains occur on OpenBookQA (+3.00), ARC-Easy (+2.61), and PIQA (+2.42), supporting concept-level capacity allocation for commonsense and question answering. Excluding the four near-chance tasks (CommonsenseQA, MMLU, C-Eval, CMMLU), the mean rises from 49.93% to 51.10%, a gain of 1.16 points.

Table 1: **Main model configurations.** Both use 1T training tokens, vocabulary size 128,815, and context length 8,192. DLCM uses the Global Parser with $R = 4$.

	Dense baseline	DLCM
Total parameters	1.3B	2.3B
Token width d_{tok}	1,536	1,536
Concept width d_{con}	—	3,072
Layers: encoder / concept / decoder	—	10 / 16 / 6
Total Transformer layers	32	32
Attention heads: token / concept	24 / —	24 / 48
KV heads: token / concept	24 / —	12 / 24
Intermediate size (self-attention blocks)	4,096	6,144
Intermediate size (cross-attention blocks)	—	6,144

Table 2: **Zero-shot accuracy on twelve benchmarks.** Scores are percentages; Δ is DLCM minus baseline in percentage points. Means are unweighted. Both models train on 1T tokens (Table 1). MMLU, CMMLU, C-Eval, and CommonsenseQA are within three points of chance for both models.

Benchmark	Dense 1.3B	DLCM 2.3B	Δ (pp)
CommonsenseQA	19.74	**21.38**	+1.64
HellaSwag	45.99	**46.66**	+0.67
WinoGrande	56.20	**57.22**	+1.02
OpenBookQA	23.80	**26.80**	+3.00
PIQA	73.10	**75.52**	+2.42
ARC-Challenge	33.04	**34.81**	+1.77
ARC-Easy	67.30	**69.91**	+2.61
MMLU	**25.70**	25.40	-0.30
BoolQ	**64.01**	62.54	-1.47
RACE	**36.03**	35.31	-0.72
C-Eval	24.37	**26.08**	+1.71
CMMLU	**25.47**	25.23	-0.24
Mean	41.23	**42.24**	**+1.01**

Knowledge and text understanding show mixed changes: C-Eval gains 1.71 points, while MMLU, CMMLU, BoolQ, and RACE decline. The benefits therefore depend on the task; these scores alone do not identify information density as the cause of individual gains or regressions.

5.3 Compression control and content adaptation

Table 3 compares Global with per-sequence regularization (*Local*), both 1.3B-parameter models trained on 1T tokens with target $R = 4$. Global achieves a realized ratio of 3.92 versus 3.15 and improves five of six scores, raising the mean by 0.72 points. Capacity is matched, so the comparison isolates the scope of the ratio loss; realized compression is part of its effect.

Granularity varies across domains: at target $R = 4$, technical English, technical Chinese, and code average 3.85, 3.27, and 3.66 tokens per concept. Appendix F gives all domains and targets. These averages describe segmentation, without defining a universal complexity ranking.

5.4 Where token prediction changes

The token-loss diagnostic uses separate 100B-token runs at the nominal 1.3B backbone scale and 600 validation samples. Figure 6 groups losses by offset from concept start: losses are lower near starts, mixed at intermediate positions, and lower at some larger offsets. A mean length near four permits segments reaching offset nineteen; later offsets sample only longer segments and do not align their right boundaries. This profile is also inconsistent with future-token leakage (Appendix B.2).

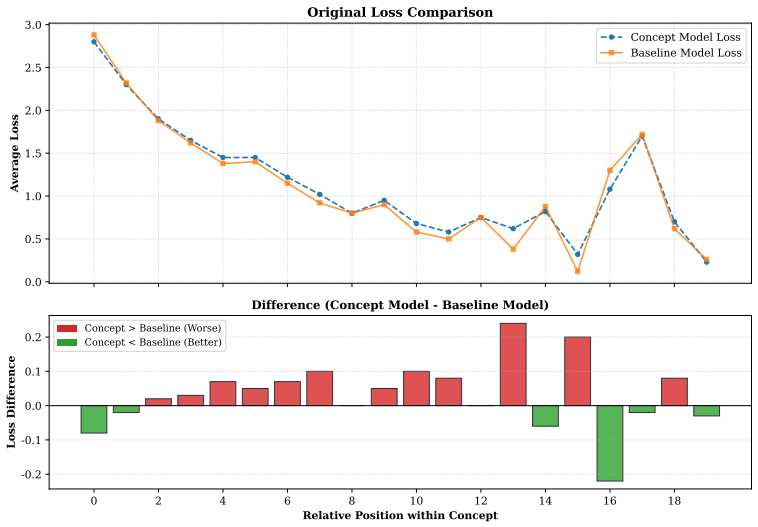

Figure 6: **Loss differences vary within concepts.** Top: token loss by offset from concept start. Bottom: concept-model minus baseline loss. The separate 100B-token comparison uses 600 validation samples. Later offsets include only sufficiently long segments and do not align right boundaries.

Table 3: **Global versus per-sequence compression regularization.** Both models have 1.3B parameters, use 1T training tokens, and target $R = 4$. Scores are zero-shot accuracy (acc) in percent. Global improves five of six scores while keeping the realized ratio closer to the target.

Benchmark	Local	Global	Δ (pp)
ARC-Challenge	28.58	**30.38**	+1.80
ARC-Easy	62.42	**62.96**	+0.54
CommonsenseQA	22.28	**24.57**	+2.29
HellaSwag	34.99	**35.07**	+0.08
OpenBookQA	**32.80**	32.20	-0.60
PIQA	67.85	**68.06**	+0.21
Mean accuracy	41.49	**42.21**	+0.72
Realized tokens/concept $\widehat{R}$	3.15	3.92	—

5.5 Boundary Stability and Kernel Performance

At input length 8,192, the learned predictor's compressed sequence grows to approximately 4,300 positions, while the cosine rule settles near 2,000 (Appendix G). Replication with FlashAttention yields 1.26–1.73× attention-kernel speedups over dynamic-mask FlexAttention (Appendix H); these timings concern the kernel operation, separately from model-level FLOPs.

6 Discussion and Conclusion

DLCM separates concept-level reasoning from token-level generation. Dynamic segmentation lets a larger backbone process a shorter sequence, while causal decoding preserves local context. At a shared 1T-token training budget, the 2.3B DLCM improves mean zero-shot accuracy over a 1.3B baseline by 1.01 points; its estimated forward FLOPs are 26.0% lower at a 32K context and 11.8% lower at the 8K training context. The compression-aware scaling law predicts lower loss at equal compute through capacity allocation, while decoupled μP supports unequal module widths.

The gains vary across tasks, and compression can sacrifice fine-grained distinctions. The scaling fits characterize the evaluated grid. These results support allocating computation at a learned intermediate granularity while preserving autoregressive token output.

REFERENCES

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin F. Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Patrick Collier, Alexey Gritsenko, Vighnesh Birodkar, Cristina Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetić, Dustin Tran, Thomas Kipf, Mario Lučić, Xiaohua Zhai, Daniel Keysers, Jeremiah Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters, 2023. URL https://arxiv.org/abs/2302.05442.

Xinrun Du, Zhouliang Yu, Songyang Gao, Ding Pan, Yuyang Cheng, Ziyang Ma, Ruibin Yuan, Xingwei Qu, Jiaheng Liu, Tianyu Zheng, et al. Chinese tiny llm: Pretraining a chinese-centric large language model. *arXiv preprint arXiv:2404.04167*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2405.18392.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers, 2020. URL https://arxiv.org/abs/2010.04245.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Shengding Hu, Yuge Tu, Xu Han, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Siming Huang, Tianhao Cheng, J. K. Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, Jiaheng Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models, 2025. URL https://arxiv.org/abs/2411.04905.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL https://arxiv.org/abs/2507.07955.

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

LCM team, Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R. Costa-jussà, David Dale, Hady Elsahar, Kevin Heffernan, João Maria Janeiro, Tuan Tran, Christophe Ropers, Eduardo Sánchez, Robin San Roman, Alexandre Mourachko, Safiyyah Saleem, and Holger Schwenk. Large concept models: Language modeling in a sentence representation space, 2024. URL https://arxiv.org/abs/2412.08821.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Kairong Luo, Haodong Wen, Shengding Hu, Zhenbo Sun, Zhiyuan Liu, Maosong Sun, Kaifeng Lyu, and Wenguang Chen. A multi-power law for loss curve prediction across learning rate schedules. In *International Conference on Learning Representations*, 2025.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.453. URL https://aclanthology.org/2025.acl-long.453/.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2025. URL https://arxiv.org/abs/2412.02595.

Howe Tissue, Venus Wang, and Lu Wang. Scaling law with learning rate annealing. *arXiv preprint arXiv:2408.11029*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL https://arxiv.org/abs/2203.03466.

Fan Zhou, Zengzhi Wang, Nikhil Ranjan, Zhoujun Cheng, Liping Tang, Guowei He, Zhengzhong Liu, and Eric P. Xing. Megamath: Pushing the limits of open math corpora. *arXiv preprint arXiv:2504.02807*, 2025. Preprint.

REPRODUCIBILITY STATEMENT

Section 3 specifies the architecture and causal decoding; Section 4 gives the optimization and scaling law. Appendices D.2–D.5 provide numerical coefficients, fit checks, and module-level cost accounting. The supplementary analysis includes the nine plotted training trajectories and a script that reproduces these checks without retraining. Appendix E distinguishes the experimental configurations and lists the corpus mixture; benchmark scores and kernel timings are reported individually.

ETHICS STATEMENT

DLCM shares the risks of pretrained language models, including biases and errors inherited from web, code, and mathematical training corpora. Computational savings do not imply greater reliability or safety. This study concerns model architecture and does not involve human participants.

AI USE STATEMENT

Generative AI assisted with manuscript structure, prose, mathematical exposition, interpretation of reported analyses, bibliography, and formatting. It did not produce new training runs or benchmark results. The authors remain responsible for the scientific claims and final submission.

A Global Compression Regularization

The Global Parser applies compression feedback to aggregate boundary statistics. Let $\mathcal{T}$ contain the valid tokens across the distributed batch and gradient-accumulation window. For the hard boundary indicators $b_{i,t}$ and continuous scores $p_{i,t}$ in Section 3.1, define

$$F = \frac{1}{|\mathcal{T}|} \sum_{(i,t)\in\mathcal{T}} b_{i,t}, \qquad G = \frac{1}{|\mathcal{T}|} \sum_{(i,t)\in\mathcal{T}} p_{i,t}. \tag{10}$$

The numerator sums and token counts are aggregated across devices. This token-weighted definition accommodates variable-length examples without giving a short sequence the same weight as a long one. The auxiliary objective uses the ratio-loss form of H-Net (Hwang et al., 2025), with a constant subtracted:

$$\mathcal{L}_{\text{aux}} = \frac{R}{R-1}\left[(R-1)FG + (1-F)(1-G)\right] - 1, \qquad R > 1. \tag{11}$$

Subtracting the constant does not change its gradients. Applying this objective after aggregating F and G differs from averaging separate per-sequence objectives: the former permits compensation between sequences with different boundary densities.

The feedback direction follows directly by treating the discrete statistic F as fixed:

$$\frac{\partial \mathcal{L}_{\text{aux}}}{\partial G} = \frac{R}{R-1}(RF - 1). \tag{12}$$

If $F > 1/R$, the derivative is positive and gradient descent lowers G; if $F < 1/R$, it raises G. In the calibrated case $F = G = u$,

$$\mathcal{L}_{\text{aux}}(u, u) = \frac{R^2}{R-1}\left(u - \frac{1}{R}\right)^2, \tag{13}$$

so its minimum is at boundary density $1/R$. This identity explains the target of the feedback, not a guarantee that discrete segmentations attain exactly that density. Training samples sharpened scores, inference thresholds them, and short sequences include a forced initial boundary. These effects make the realized ratio $\widehat{R}$ an empirical quantity, as reported in Tables 3 and 14.

Discrete boundary assignments determine which states are pooled. The cosine rule operates on learned representations, while the discrete assignments are decoupled from direct CE optimization. Equation 12 describes the compression signal to the continuous scores without requiring a derivative through a sampled indicator. The method therefore distinguishes learning the representations from directly learning a discrete segmentation policy through token prediction.

B Alignment Between Concepts and Token Predictions

For clarity, let s_j be the first token index of segment j. An observed boundary at s_{j+1} identifies the preceding segment as $S_j = \{s_j, \ldots, s_{j+1} - 1\}$. Its pooled state contains all encoder outputs in that interval. If this closing boundary is needed to construct c_j, then c_j is available no earlier than observation s_{j+1}. The availability index a_j in Equation 5 includes this dependency. Merely replacing it by the segment index of a query would expose information from later tokens inside the segment.

Let $j^*(t) = \max\{j : a_j \le t\}$ denote the most recent available concept at an observation step where at least one concept is available. The token query at t can use the concept prefix $\widetilde{Z}_{\le j^*(t)}$ to predict x_{t+1}. Several consecutive queries can have the same $j^*(t)$; they share concept representations while retaining distinct token states h_t. This is the sense in which a completed concept supports multiple subsequent tokens. It does not remove the conditioning on preceding generated tokens:

$$p(x_{t+1:t+k} \mid x_{\le t}) = \prod_{u=t}^{t+k-1} p(x_{u+1} \mid x_{\le u}). \tag{14}$$

In a packed training pass, the attention mask enforces the same ordering even though all token positions are present in memory. During generation, the available concept prefix grows as new concepts are formed. The causal token pathway preserves local context between concept updates.

B.1 Replication and softmax weighting

Replication turns an irregular mapping into token-aligned runs. Let $r(u)$ identify the concept assigned to replica position u. The expanded keys and values are $K'_u = K_{r(u)}$ and $V'_u = V_{r(u)}$, with unavailable concepts masked out. The implementation uses `repeat_interleave` to form these runs and evaluates attention with variable-length FlashAttention (Dao et al., 2022).

Replication shares representations but changes the normalization domain. For a fixed query, let $m_{t,j}$ be the number of visible copies of concept j, and let $s_{t,j} = q_t^\top k_j/\sqrt{d_h}$ denote its attention logit. When replicas have identical logits, grouping their contributions gives

$$u_t^{\text{rep}} = \frac{\sum_j m_{t,j} \exp(s_{t,j}) v_j}{\sum_j m_{t,j} \exp(s_{t,j})}. \tag{15}$$

Concept-level attention instead uses one copy of each visible concept. The two forms coincide when all visible multiplicities are equal, but not for arbitrary segment lengths. For example, equal logits for values 0 and 1 yield an average of $1/2$ with one copy each and $3/4$ with multiplicities 1 and 3. Thus replication is described as a token-aligned attention implementation, not as an unconditional algebraic equivalence to attention over unique concepts. Its timing results appear in Appendix H.

B.2 Causality of concept exposure

A hierarchical model can leak future information if a pooled concept is exposed to the tokens used to form it. In DLCM, concept j is visible to the query at position t only if $a_j \le t$ (Equation 5), where a_j is the position at which the boundary closing segment j is observed. In the replicated implementation (Section 3.4), concept j is replicated only at key positions $u \ge a_j$, so the standard causal kernel enforces the same condition (Figure 10). Training and generation use the same replicated layout, so this ordering holds in both.

The token-loss profile is consistent with this construction. If a concept exposed later tokens of its own segment, the model could copy them and within-concept losses would fall far below the baseline. Figure 6 instead shows differences below 0.25 nats at every offset, with the concept model slightly worse than the baseline at most within-concept offsets.

C Decoupled Maximal-Update Parametrization

The token modules and concept backbone use width multipliers $s_{\text{tok}} = d_{\text{tok}}/d_{\text{base}}$ and $s_{\text{con}} = d_{\text{con}}/d_{\text{base}}$. Here d_{base} denotes the reference width against which each group is scaled; the resulting learning rates need not be equal even when their base rates are similar. Table 4 collects the within-module rules used in the parametrization.

Table 4: **Width-dependent optimization.** The multiplier s is chosen for the token or concept module containing a hidden layer. Embeddings and biases retain their base learning rates.

Quantity	Token modules	Concept backbone
Hidden-weight variance	$\sigma^2_{\text{base}}/s_{\text{tok}}$	$\sigma^2_{\text{base}}/s_{\text{con}}$
Hidden-layer learning rate	$\eta^{\text{base}}_{\text{tok}}/s_{\text{tok}}$	$\eta^{\text{base}}_{\text{con}}/s_{\text{con}}$
AdamW epsilon	$\epsilon_{\text{base}}/s_{\text{tok}}$	$\epsilon_{\text{base}}/s_{\text{con}}$

Embedding initialization variance is fixed at σ^2_{base}, and output logits use the token multiplier:

$$\text{logits} = s_{\text{tok}}^{-1} h_{\text{final}} W_{\text{unemb}}^\top. \tag{16}$$

The tuning procedure follows the proxy-transfer approach of Yang et al. (2022). Coordinate sweeps on an 87M model vary a base learning rate by factors $\{0.5, 0.75, 1.5, 2.0\}$ relative to the current reference setting. The selected token and concept base rates are then transferred to models of approximately 274M, 468M, and 834M parameters. The legends `xxs`, `xs`, `s`, and `base` in Figure 2 refer to these four scales in increasing order.

The joint perturbation plot varies both base rates by the same multiplicative factor. Its shared minimum near the reference setting supports transfer over the evaluated scales. This is an empirical

result for the heterogeneous architecture; it does not establish a general equivalence between every possible token/concept interface and a uniform-width Transformer.

D Compression-Aware Scaling Formulation

Equation 7 separates the loss contributions associated with token capacity, concept capacity, and data. Table 5 defines its quantities. The parameter partition treats the concept backbone as NP and its complement as $N(1-P)$; token processing and interface capacity belong to that complement. The offsets have the same units as the corresponding parameter or token counts and represent finite-size shifts in the fitted denominators.

Table 5: **Scaling-law notation.** Capacity and data exponents describe the fitted loss dependence; the compression exponent is separate from the FLOPs cost model.

Symbol	Meaning
N, P	Total parameters and fraction allocated to the concept backbone
$N(1-P), NP$	Token/interface capacity and concept-backbone capacity
D	Training tokens consumed at the observation checkpoint
R	Target average number of tokens per concept
E_0	Fitted loss floor
$A_{\text{tok}}, A_{\text{con}}, A_{\text{data}}$	Amplitudes of the three loss contributions
$\delta_{\text{tok}}, \delta_{\text{con}}, \alpha$	Token-capacity, concept-capacity, and data exponents
γ	Compression exponent in the concept-capacity term
$t_{\text{tok}}, t_{\text{con}}, t_{\text{data}}$	Finite-size offsets in parameter or token units

D.1 Relation to a single-capacity scaling law

At $R = 1$, the explicit factor R^{γ} becomes one. Two capacity terms remain, so the law does not generally reduce to a single power of total parameter count. A special case makes the relationship precise. If the capacity offsets vanish, P is fixed, and $\delta_{\text{tok}} = \delta_{\text{con}} = \delta$, then

$$\frac{A_{\text{tok}}}{[N(1-P)]^{\delta}} + \frac{A_{\text{con}}}{(NP)^{\delta}} = \frac{A_{\text{eff}}(P)}{N^{\delta}}, \quad A_{\text{eff}}(P) = \frac{A_{\text{tok}}}{(1-P)^{\delta}} + \frac{A_{\text{con}}}{P^{\delta}}. \tag{17}$$

If the data offset also vanishes, this yields the familiar capacity–data form $E_0 + A_{\text{eff}}/N^{\delta} + A_{\text{data}}/D^{\alpha}$ (Hoffmann et al., 2022). The general DLCM form retains the two capacities because allocating parameters between them is itself a design variable. This $R = 1$ comparison concerns the loss model; the compression-control objective in Equation 11 is defined for $R > 1$.

D.2 Trajectory and decay fits

The nine trajectories underlying Figures 3 and 4 cross three sizes with $R \in \{2, 4, 8\}$. Each contains 40,000 steps and reaches $D_{\max} = 167{,}772{,}160{,}000$ tokens. Table 6 gives the logged total and backbone parameter counts. In this subset, P is fixed within each size; it is not an independent three-value sweep at each size. The saved stable-phase coefficients, including the offsets, are shared across all nine trajectories.

Table 6: **Configurations of the plotted trajectories.** Each row has runs at $R = 2, 4, 8$, all reaching 167.77B tokens. $P = N_{\text{con}}/N$ includes vocabulary parameters in the denominator.

Plot label	Total parameters	Backbone parameters	P
274M	274,434,049	86,930,177	0.316762
468M	468,201,729	154,703,617	0.330421
833M	833,813,761	347,659,777	0.416951

Table 7 supplies the numerical loss-law coefficients. Capacity and data arguments are in billions: substitute $N/10^9$ and $D/10^9$ for the raw counts in Equation 7. The offsets use these same units.

Table 7: **Saved scaling coefficients.** Stable-trajectory values are rounded here; the supplementary analysis preserves their stored precision. Exponentials in the analytical column convert the saved log amplitudes.

Quantity	Stable trajectories	Analytical projection
E_0	2.11652422	$e^{0.376}$
A_{tok}	0.05907163	$e^{-1.061}$
t_{tok}	0.48851699	0.017
δ_{tok}	3.68068981	0.345
A_{con}	0.35611457	$e^{-3.010}$
t_{con}	0.00500023	0.060
δ_{con}	0.13111109	0.472
γ	0.07769037	0.449
A_{data}	15.29405975	$e^{2.391}$
t_{data}	1.25999653	0.507
α	1.49385095	0.914

The first column reconstructs the stable-phase lines; the second belongs to the separate analytical projections. They are distinct parameter sets.

For Figure 3, losses are filtered to $(0.01, 20)$, sorted by tokens, and smoothed with a trailing 100-step arithmetic mean. Evaluating the stored coefficients over $D \leq 0.9D_{\max}$ gives 324,000 observations, pooled $R^2 = 0.9875$, and RMSE 0.0766. Restricting this check further to $D \geq 10$B gives $R^2 = 0.8888$ and RMSE 0.0500. The pooled statistic is therefore sensitive to the large loss range early in training. Consecutive observations within a trajectory are strongly autocorrelated, so the effective sample size is closer to the nine runs than to the number of logged steps; we therefore emphasise the $D \geq 10$B statistics and the endpoint errors. These checks reuse the plotted training trajectories and do not estimate out-of-sample accuracy or coefficient uncertainty.

The stars in Figure 3 apply a simplified endpoint correction to the stable-law prediction. Figure 4 instead uses a robust correction with explicit allocation terms. Both can be written as

$$\Delta_{\text{decay}} = k\, L_{\text{stable}}^{a} R^{b} N_{\text{raw}}^{c} P^{d_1} \exp\left[d_2 (\log P)^2\right], \tag{18}$$

where N_{raw} is the unscaled parameter count and $L_{\text{final}} = L_{\text{stable}}(1 - \Delta_{\text{decay}})$. Table 8 distinguishes the saved corrections. The simplified correction uses $d_1 = d_2 = 0$ and clips the fractional reduction to $[0, 0.99]$.

Table 8: **Saved decay coefficients.** The first two columns correspond to the two distinct trajectory figures. The last is used only in the analytical savings sweep. All use raw parameter counts.

	Simplified stars	Robust endpoints	Savings sweep
k	2.18896249×10^{5}	0.0039581035	$1.08754821 \times 10^{-15}$
a	−6.13423532	−0.01133990	7.86566464
b	0.04717897	−0.02807893	−0.12744852
c	−0.48581958	0.09593610	1.09359156
d_1	0	−0.03816403	−1.21970812
d_2	0	0.07050758	0

For the robust correction, training losses are smoothed by exponentiating a centered 1,000-step rolling mean of log loss. L_{stable} averages this curve over $0.89D_{\max} \leq D \leq 0.90D_{\max}$; L_{final} averages it over $D \geq 0.99D_{\max}$. This centered smoothing is descriptive and uses neighboring observations. The saved fitting procedure minimizes mean Huber loss on log fractional reductions, with threshold 0.5 and an ℓ_2 penalty of 0.01 on the five exponents. It selects among 81 L-BFGS-B initializations, each limited to 2,000 iterations; the intercept is unpenalized.

Table 9 evaluates the saved robust coefficients on the nine reductions. The R^2 of log fractional reduction is 0.5515; final-loss RMSE is 0.00279 and the largest absolute endpoint error is 0.00417. Endpoint errors and reduction-level R^2 describe different quantities and should not be interchanged. No confidence interval follows from these single-trajectory checks.

Table 9: **Recomputed robust-decay endpoints.** Observed and predicted loss use the preprocessing above. The error column is prediction minus observation.

Size	R	Observed final loss	Predicted final loss	Error
274M	2	2.790749	2.789269	−0.001480
274M	4	2.824169	2.823550	−0.000618
274M	8	2.837192	2.833022	−0.004170
468M	2	2.636534	2.640122	+0.003587
468M	4	2.663873	2.666331	+0.002458
468M	8	2.683421	2.687471	+0.004050
833M	2	2.514006	2.510793	−0.003213
833M	4	2.544801	2.546250	+0.001449
833M	8	2.570173	2.568625	−0.001548

D.3 Analytical allocation and FLOPs sweeps

Figure 7(a) fixes a target loss before comparing compute. The analytical baseline fit at $N = 1.3$B and $D = 1$T supplies $L_{\text{target}} = 2.056583$. For each $R \in \{2, 4, 8\}$ and each of 40 evenly spaced backbone shares between 10% and 99%, the calculation solves for the size achieving that target. Thus the plotted quotient is $L_{\text{target}}/\mathcal{C}_{\text{lin}}$: its numerator is constant, and maximizing it minimizes compute within this grid. Recomputing the sweep yields $P = 0.373846$, $R = 2$, $N = 0.660587$B, and $\mathcal{C}_{\text{lin}} = 3.22265 \times 10^{21}$ FLOPs. This explains the rounded 37.4% setting in Figure 8; it establishes a grid optimum at this target loss, not a global optimum at every budget.

Panel (b) uses $P = 0.5$, $R = 4$ (denoted C), a rounded version of the stable-trajectory coefficients, and the separate savings-sweep decay coefficients in Table 8. Its baseline surrogate evaluates that stable law at $P = 0, R = 1$ and applies a fixed 11.98% reduction. Hierarchical reductions are clipped to $[0, 0.99]$, and model size is solved at equal predicted final loss. Its annotated 40.2% saving near a 1.3B baseline belongs to this analytical comparison. It is separate from the 2.3B-versus-1.3B main-model inference estimate.

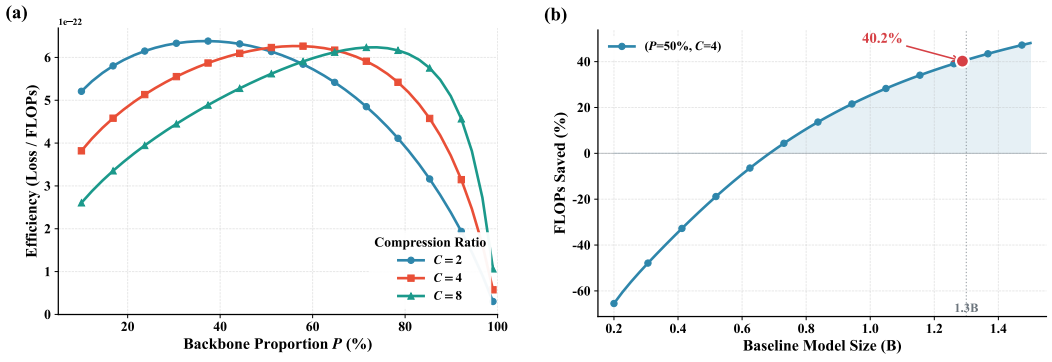

Figure 7: **Analytical capacity-allocation sweeps.** Left: a fixed target loss divided by estimated training FLOPs, so higher values mean lower compute at equal predicted loss. Right: a separate equal-loss savings sweep at $P = 50\%$, $R = 4$. Both use the parameter-linear cost proxy and are distinct from the main-model inference estimate.

D.4 Projection coefficients and matched-budget accounting

The analytical projection column of Table 7 is paired with the dense-model law

$$\widehat{L}_{\text{dense}}(n, d) = 1.6890 + \frac{0.3973}{(n + 0.0100)^{0.3013}} + \frac{53.8974}{(d + 2.8506)^{1.5356}}, \tag{19}$$

where $n = N/10^9$ and $d = D/10^9$. This dense fit is distinct from the baseline surrogate used in the savings sweep. The full-trajectory and analytical-projection coefficients are also distinct; fit quality on the nine trajectories does not validate the analytical projection at larger sizes.

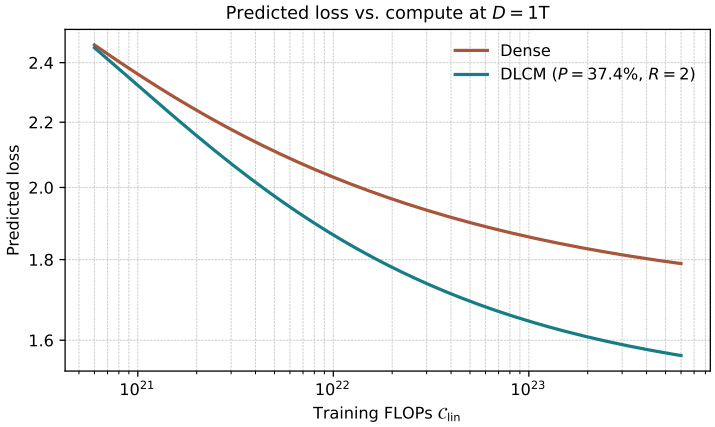

Figure 8: **Fixed-configuration loss–compute projection.** Predicted loss versus training FLOPs ($\mathcal{C}_{\text{lin}}$) at $D = 1$T for the dense model and DLCM with $P = 37.4\%$, $R = 2$. These projections are separate from the main $R = 4$ benchmarks.

Two cost conventions are in common use for the concept backbone:

$$\mathcal{C}_{\text{lin}} = 6DN\left[(1-P) + P/R\right], \tag{20}$$

$$\mathcal{C}_{\text{quad}} = 6DN\left[(1-P) + P/R^2\right]. \tag{21}$$

Here N, D are raw counts. At $P = 0.374$, $R = 2$, their respective cost factors are 0.813 and 0.7195. The sequence-squared approximation extends attention-pair scaling to the entire concept-capacity term, undercounting parameter-linear operations. The linear convention follows how projections and feed-forward work depend on concept positions, while still omitting attention-pair and interface overheads.

Table 10 recomputes the comparison at exactly matched budgets with the parameter-linear convention. For a budget B, the dense size is $B/(6D)$ and the hierarchical size is $B/[6D(1-P+P/R)]$. Both losses are evaluated at $D = 1$T, keeping $P = 0.374$, $R = 2$ fixed. The hierarchical projection remains lower under this convention; these values are model-based extrapolations, not additional measured training runs or an optimized frontier.

Table 10: **Matched-budget predictions with parameter-linear concept cost.** Sizes are in billions of parameters. All rows use $D = 1$T and the analytical coefficients.

Budget (FLOPs)	Dense size	DLCM size	Dense loss	DLCM loss
10^{21}	0.166667	0.205002	2.360136	2.321658
10^{22}	1.666667	2.050021	2.030338	1.865968
10^{23}	16.666667	20.500205	1.860503	1.645289

D.5 Module-level forward FLOPs

We count one multiply-add as two FLOPs and separate dense projections, attention products, and feed-forward computation. For a self-attention block with n positions, width d, key/value-to-query head ratio g, and K visible query–key pairs, the projection and attention-product costs are

$$\mathcal{F}_{\text{proj}} = 4nd^2(1+g), \qquad \mathcal{F}_{\text{attn}} = 4dK. \tag{22}$$

The head dimension is d/H_q, with $g = H_{kv}/H_q$. A causal full-sequence pass has $K = n(n+1)/2$; a dense rectangular calculation that also evaluates masked entries has $K = n^2$. Grouped-query attention reduces key/value projection work, but all query heads still contribute attention products. A feed-forward block with intermediate width f adds $4ndf$ FLOPs for two matrices or $6ndf$ for

Table 11: **Complete forward-FLOPs accounting.** Batch size one, $L = 8{,}192$, $M = L/4$, exact causal pair counts, replicated concept keys/values in the decoder (as in training and generation); units are 10^{12} FLOPs. Elementwise operations, normalization, and the smoothing module are excluded. Under this convention DLCM uses 11.8% fewer forward FLOPs than the dense baseline; Figure 9 reports other context lengths and the rectangular convention.

Model	Module	FLOPs (10^{12})
Dense	Self-attention projections	4.948
Dense	Self-attention products	6.598
Dense	Feed-forward	9.896
Dense	Vocabulary head (all positions)	3.242
Dense	**Total**	**24.683**
DLCM	Encoder attention projections	1.160
DLCM	Encoder attention products	2.062
DLCM	Encoder feed-forward	4.639
DLCM	Boundary and pooling projections	0.097
DLCM	Backbone attention projections	1.855
DLCM	Backbone attention products	0.413
DLCM	Backbone feed-forward	3.711
DLCM	Decoder cross-attention projections	0.580
DLCM	Decoder cross-attention products	1.237
DLCM	Decoder feed-forward	2.783
DLCM	Vocabulary head (all positions)	3.242
DLCM	**Total**	**21.777**

a three-matrix gated form. Elementwise nonlinearities, normalization, positional operations, and memory movement are outside these matrix-operation counts.

Let L and M be the observed token and concept lengths. Encoder costs use $n = L, d = d_{\text{tok}}$; backbone costs use $n = M, d = d_{\text{con}}$. Thus their projection and feed-forward terms decrease linearly with M, while their full-sequence attention products depend on M^2. Compression does not divide every backbone operation by R^2.

For cross-attention with token queries, unique concept keys/values, attention width d_a, and key/value ratio g_x, the dense projection cost is

$$\mathcal{F}_{\text{cross,proj}} = 4Ld_{\text{tok}}d_a + 4Md_{\text{con}}g_x d_a, \qquad \mathcal{F}_{\text{cross,attn}} = 4d_a \sum_{t=1}^{L} \#\{j : a_j \le t\}. \tag{23}$$

These expressions count one key/value projection per concept. Replicated attention replaces the number of visible concepts by the number of visible replicas for its attention products. If key/value projection occurs after replication, its projection term must also use the replicated length. Consequently, the precise decoder layout, ordering of projection and replication, and causal mask are part of a complete model-level count.

The vocabulary projection adds $2Jd_{\text{tok}}V$ FLOPs for J positions whose logits are evaluated; $J = L$ for all-position scoring and $J = 1$ for a single next-token distribution. Lookup embeddings themselves are memory operations. Boundary projections add $4Ld_{\text{tok}}d_{\text{scan}}$, and the pooling projection adds $2Md_{\text{tok}}d_{\text{con}}$; smoothing and any further interfaces contribute their own costs. Summing these terms with the actual module multiplicities gives Equation 8.

Table 11 applies these expressions to every module of both models, using the widths, head counts, and intermediate sizes in Table 1 and $M = L/4$. Its totals give the savings quoted in the main text; the accounting script is included in the supplementary material. Figure 9 repeats the calculation for other context lengths, for the rectangular convention ($K = n^2$), and for a decoder that keeps one key/value per concept, which would raise the saving to 15.5% at 8K and 34.3% at 32K.

For cached generation, a token module performs its projections and feed-forward work at every token, while the concept backbone updates only when a new concept becomes available. The amortized backbone cost is the sum of those update costs divided by generated token count (Figure 5). Each update attends to its existing concept cache; the full-sequence quadratic pair factor is therefore

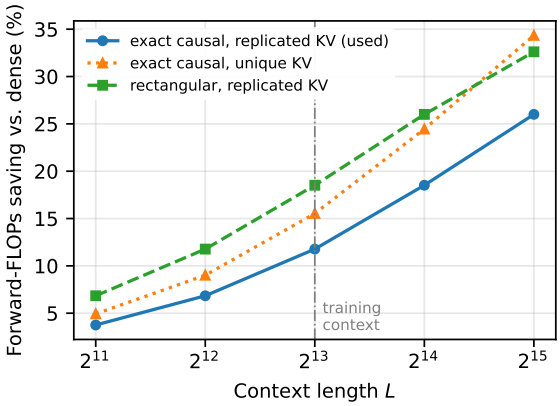

Figure 9: **Inference-FLOPs saving grows with context length.** Forward FLOPs of the 2.3B DLCM relative to the 1.3B dense baseline, computed module by module (Table 11) under three conventions. Under exact causal counting the saving is 11.8% at 8K and 26.0% at 32K. Both models are trained with an 8K context (dashed line); longer contexts are extrapolations of the cost model, not measured runs.

not a per-token decoding-speed factor. Training FLOPs additionally include backward computation and any recomputation, so the $6DN$ proxy, forward matrix counts, and measured kernel times describe different costs.

E Data and Experimental Settings

The main pretraining mixture draws from the open-source corpora in Table 12. The total is one trillion tokens after allocation across the four sources. A mixture assembled from public corpora is distinct from a release of the exact processed training stream; the table specifies its composition without implying an additional dataset release.

Table 12: **Pretraining mixture.** Percentages and token allocations sum to 100% and 1T tokens, respectively.

Source	Content	Share	Tokens
Nemotron-CC (Su et al., 2025)	English web	50%	500B
MAP-CC (Du et al., 2024)	Chinese web	25%	250B
OpenCoder (Huang et al., 2025)	Code	15%	150B
MegaMath (Zhou et al., 2025)	Mathematics	10%	100B
Total		100%	1T

Table 13 distinguishes the model sizes and training budgets of the analyses. The main comparison (2.3B) and the Global Parser comparison (1.3B) use different DLCM models, while the within-concept loss diagnostic uses a separate shorter training run. This distinction is necessary when interpreting their absolute scores.

The main models use the DeepSeek-V3 tokenizer (Liu et al., 2024), vocabulary size 128,815, context length 8,192, and Swish activations. Table 1 gives their widths, attention heads, and layer allocation. The baseline feed-forward intermediate width is 4,096; the DLCM configuration lists intermediate widths of 6,144 for self- and cross-attention blocks. These main-model settings should not be applied to the smaller parser model or the scaling proxies.

Table 13: **Experimental blocks.** Parameter counts refer to total model size unless the row explicitly identifies a backbone scale. The parser comparison matches capacity; realized compression differs.

Analysis	Model configuration	Training tokens
Main zero-shot results	DLCM 2.3B; dense baseline 1.3B	1T each
Parser comparison	Global Parser 1.3B; Local 1.3B	1T each
Token-loss profile	Concept and baseline models at nominal 1.3B backbone scale; 600 validation samples	100B each
μP transfer	87M proxy; 274M, 468M, approximately 834M transfer models	Proxy/transfer runs
Plotted scaling trajectories	274M, 468M, approximately 834M; logged P in Table 6; $R \in \{2, 4, 8\}$	167.77B each

F Content-Dependent Segmentation

Table 14 reports realized average tokens per concept across six content types. A target ratio constrains an aggregate statistic; it does not prescribe each domain's mean or the length of an individual segment. The changing order across columns also shows why these statistics should not be treated as a fixed ranking of semantic complexity.

Table 14: **Average tokens per concept.** Columns identify training targets; entries are observed domain-level averages.

Content	Target $R = 8$	Target $R = 4$	Target $R = 2$
Casual English	7.47	3.53	1.76
Casual Chinese	8.38	4.36	1.76
Technical English	10.58	3.85	1.92
Technical Chinese	6.09	3.27	1.76
Code	6.14	3.66	1.98
Math/Science	7.42	4.41	1.91

The examples below use the same text span across all three ratios within each domain. They are common-prefix excerpts of the recorded segmentations, with vertical bars marking the observed boundaries. Counts for the longer source passages are not assigned to these excerpts. Boundaries can split a contraction or identifier because concepts are operational latent segments, not guaranteed linguistic constituents. The tokenizer defines the available token boundaries; these examples do not compare alternative tokenizers.

F.1 Casual English

Target $R = 8$.

> So I | 've been trying to perfect my morning coffee routine lately. | It's funny how something | so simple can have so many variables.

Target $R = 4$.

> So I | 've been trying | to perfect | my morning coffee routine lately. | It's | funny how something | so simple can | have so many variables.

Target $R = 2$.

> So I | 've | been trying to | perfect | my | morning coffee routine | lately. | It | 's funny how | something so | simple | can | have so many | variables.

F.2 Python code

Target $R = 8$.

```
import torch | from torch.utils.data import Dataset,
DataLoader |
class SimpleTextDataset(Dataset):
```

Target $R = 4$.

```
import torch | from torch.utils.data | import Dataset | ,
DataLoader |
class Simple | TextDataset(Dataset):
```

Target $R = 2$.

```
import | torch | from torch.utils.data import | Dataset, |
DataLoader |
class | Simple | TextDataset(Dataset):
```

F.3 Mathematical exposition

Target $R = 8$.

> Euler's formula is a | mathematical formula in complex analysis that establishes | the fundamental relationship between the trigonometric functions and the complex exponential function. | The formula states that | for any real number x, e^{ix} = | $\cos(x) + i\sin(x)$,

Target $R = 4$.

> Euler's | formula is | a mathematical formula in | complex analysis that establishes | the fundamental relationship between | the trigonometric functions and the | complex exponential function. | The formula states | that for any | real number x | , e^{ix} = | $\cos(x) + i\sin(x)$,

Target $R = 2$.

> Euler | 's | formula is | a | mathematical formula | in | complex analysis that | establishes | the fundamental relationship between | the trigonometric functions | and | the complex exponential | function. | The | formula states | that | for | any | real number x | , | e^{ix} = | $\cos(x)$ | + | $i\sin(x)$,

G Boundary-Predictor Behavior

The stability comparison contrasts the final cosine-based rule with a separately learned discrete boundary predictor. For an 8,192-token input, the learned predictor's compressed length increases during training and reaches approximately 4,300 positions, corresponding to about 1.9 tokens per segment. The cosine-rule trajectory settles near 2,000 positions, or about four tokens per segment. These approximate values summarize the observed trajectories, not repeated-run means.

Reducing compression preserves more token detail and can improve the token-prediction objective, which motivates separating the discrete boundary decision from its direct CE optimization. The comparison supports that design choice for the evaluated predictors. It does not establish that every learned boundary policy is unstable, nor does it equate a stable compression ratio with a uniquely correct linguistic segmentation.

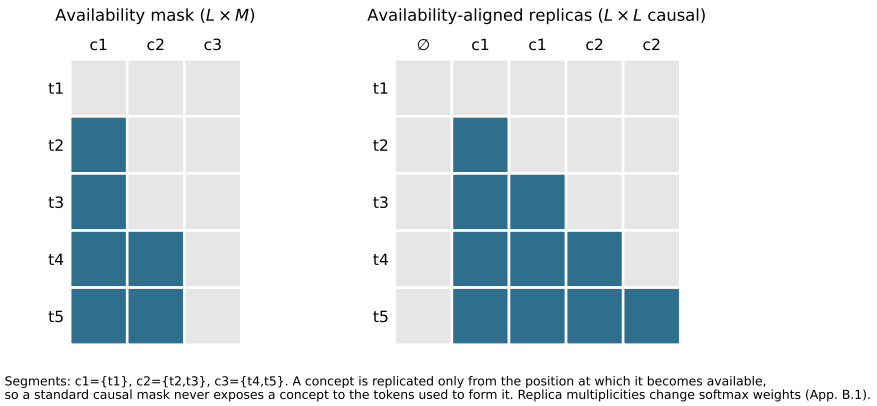

Figure 10: **Availability-aligned replication.** Left: the token-to-concept mask of Equation 5 for segments $c_1 = \{t_1\}, c_2 = \{t_2, t_3\}, c_3 = \{t_4, t_5\}$; a concept becomes visible only after the boundary that closes it is observed. Right: each concept is replicated from its availability position onward, so a standard causal kernel reproduces the same visibility without a custom mask. Replica multiplicities change softmax weights (Appendix B.1).

H Attention-Kernel Measurements

Table 15 compares dynamic-mask FlexAttention with the replication/FlashAttention implementation described in Section 3.4. The recorded configuration uses batch size one and 32 heads. Speedup is calculated as $T_{\text{Flex}}/T_{\text{Flash}}$. The measurements characterize this attention workload; they are separate from the model-level FLOPs estimate and do not measure full-model generation latency.

Table 15: **Attention-kernel timings.** Times are in milliseconds. All twelve measurements use the same batch size and head count.

Sequence length	Hidden width	Flex	Flash Varlen	Speedup
2,048	1,024	32.35	22.48	1.44×
2,048	2,048	33.31	22.58	1.48×
2,048	4,096	32.42	22.56	1.44×
4,096	1,024	59.75	45.15	1.32×
4,096	2,048	60.72	45.17	1.34×
4,096	4,096	65.88	45.48	1.45×
8,192	1,024	116.35	91.42	1.27×
8,192	2,048	114.65	90.66	1.26×
8,192	4,096	142.75	96.79	1.47×
16,384	1,024	314.35	186.21	1.69×
16,384	2,048	323.53	186.83	1.73×
16,384	4,096	315.69	190.38	1.66×