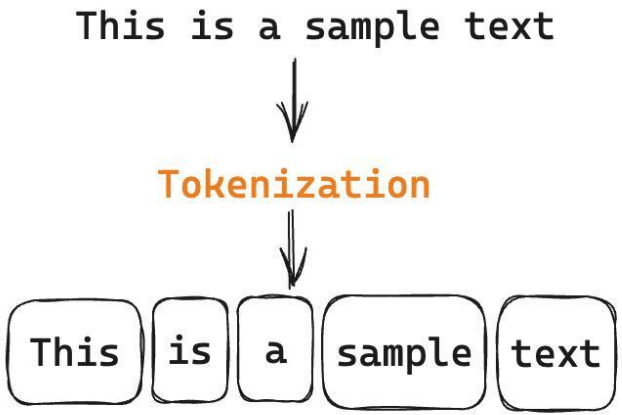
Can dynamic byte-level patches outperform fixed tokens as the basic unit for scaling LMs?

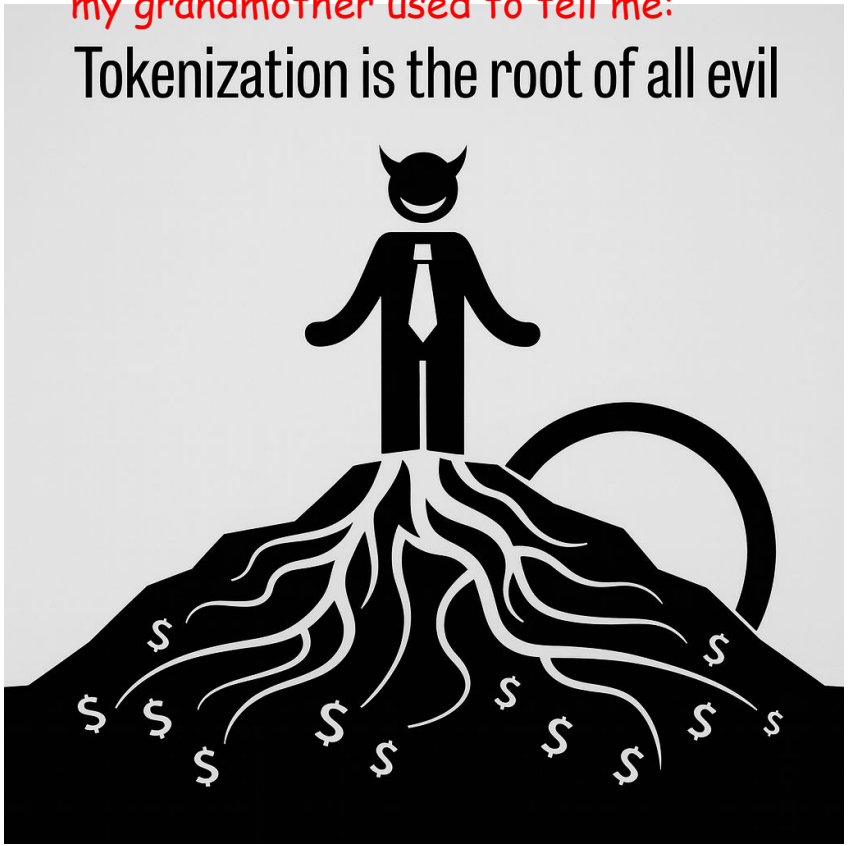
Gus Fridell, Ernie Chu, Alex Martin, Austen Liao, Kuleen Sasse

Tokenization Intro

- Tokenization is how we discretize language input and prepare it for model evaluation



my grandmother used to tell me:



- Biased
- Heuristic preprocessing step
- Leads to input sensitivity
- Lack of spelling knowledge
- Domain/modality sensitivity
- Multilingual inequity
- Static amount of compute per token
- …

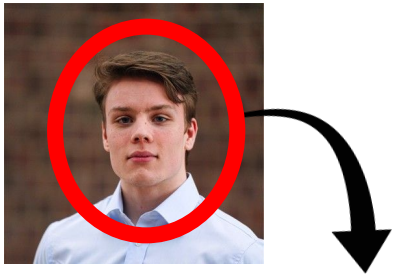
Does tokenization belie the “spirit” of deep learning?

- End-to-end learning philosophy: learn hierarchical representations from raw data
- Tokenization is a fixed processing step between raw text and the model
  - static, context-independent boundaries
- ”The Bitter Lesson”
  - "The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin."

Methods of Tokenization

Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin

Methods of Tokenization



Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin

Methods of Tokenization: Characters

Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin

D, a, e, n, e, r, y, s, ‘’, T, a, r, g, a, r, y, e, n… (I got tired)

Methods of Tokenization: BPE

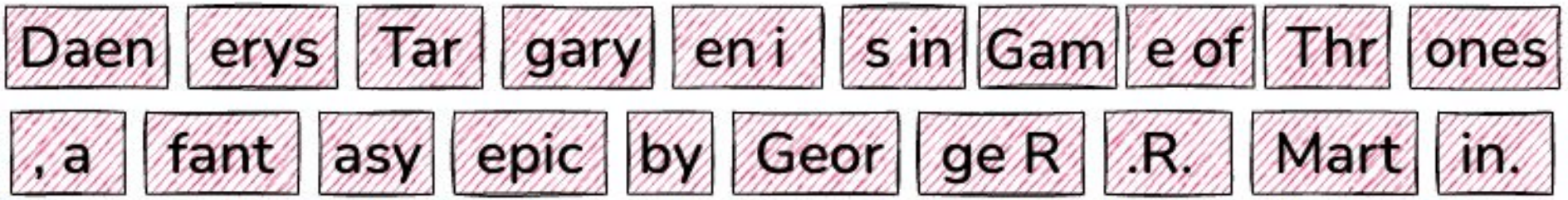
Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin

D, a, e, n, e, r, y, s, ‘’, T, a, r, g, a, r, y, e, n… (Character level)

Subword merge

Methods of Tokenization: Strides (4)

Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin

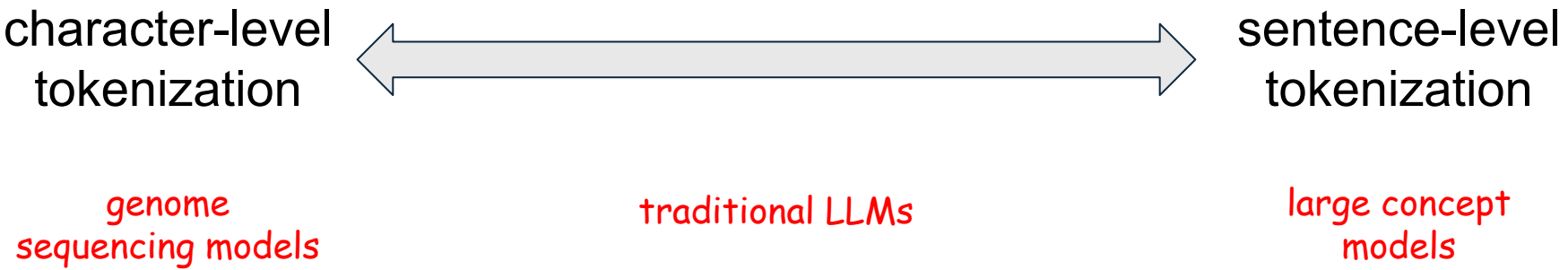


Methods of Tokenization: Space

Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin

Methods of Tokenization: Tradeoff

- smaller tokens: richer representation but longer sequences (and vice versa)
- different tokenization schemes suited for different sequencing tasks



Byte Latent Transformer

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, et al.

Motivation

Motivation

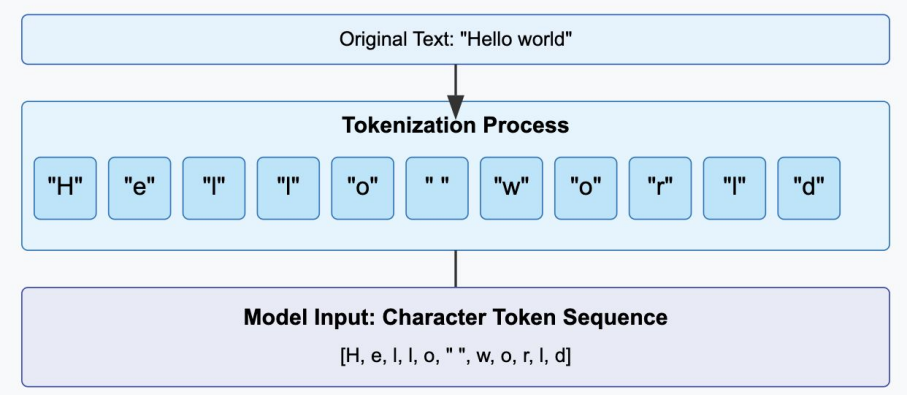
- Biased
- Heuristic preprocessing step
- Leads to input sensitivity
- Lack of spelling knowledge
- Domain/modality sensitivity
- Multilingual inequity
- Static amount of compute per token
- …

Motivation

Can we match tokenizers without using tokenization at all?

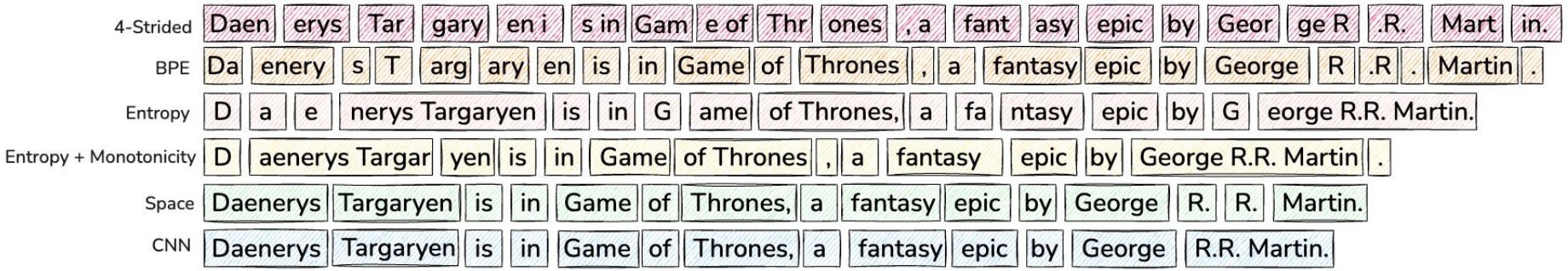
Background: Character Transformers

- Based on representing text as a sequence of individual characters
- Pros:
  - Flexible (no OOV)
  - Dealing with morphologically rich languages
  - Robust to noise
  - Small vocabulary
- Cons:
  - Creates long sequence lengths
  - More expensive to train



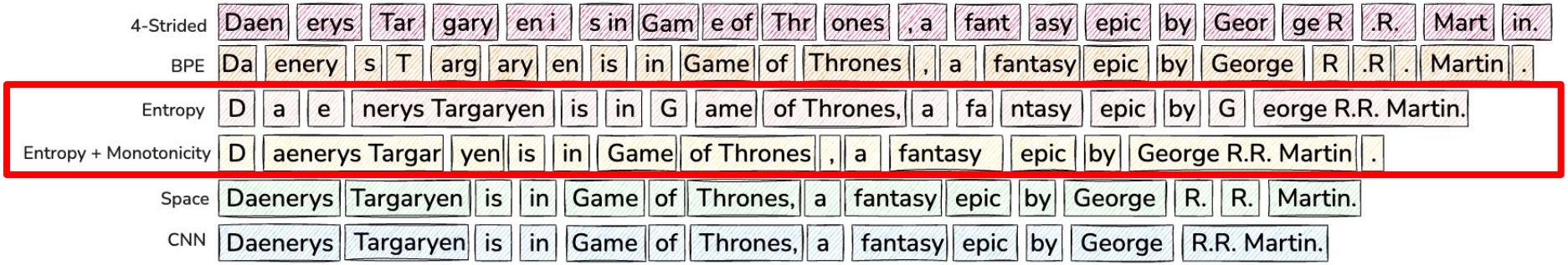
Patching

- The way the model dynamically groups the bytes into chunks (without vocab)
- Allows BLT to dynamically allocate compute based on context
- Many different methods:



Patching

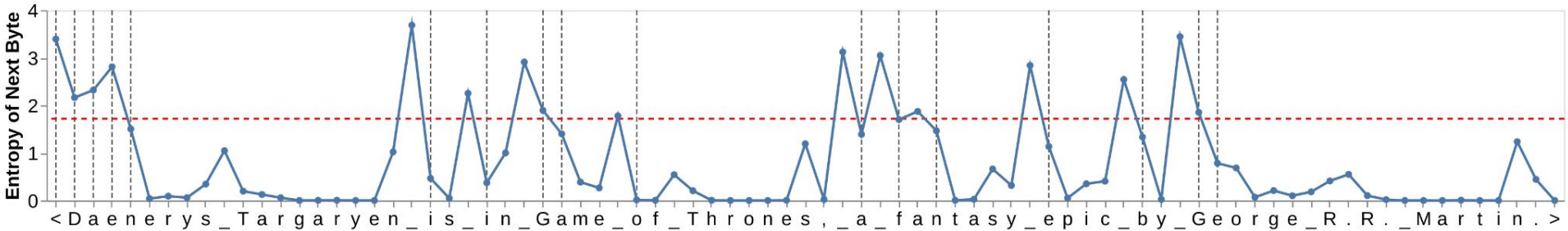
- The way the model dynamically groups the bytes into chunks (without vocab)
- Allows BLT to dynamically allocate compute based on context
- Many different methods:



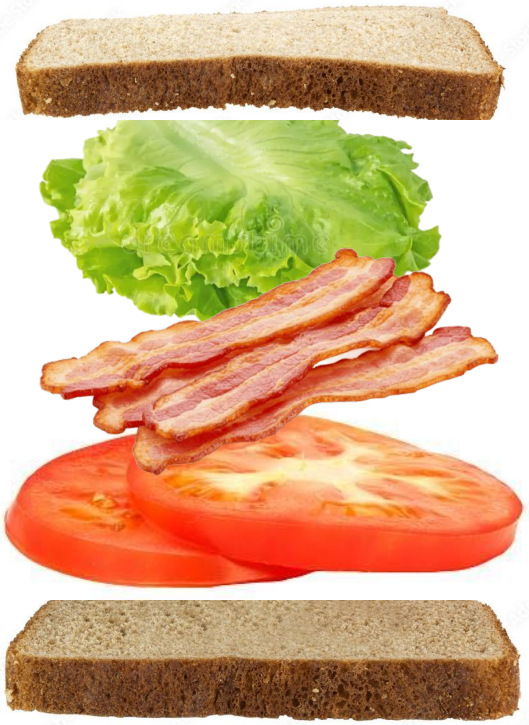
Entropy Patching

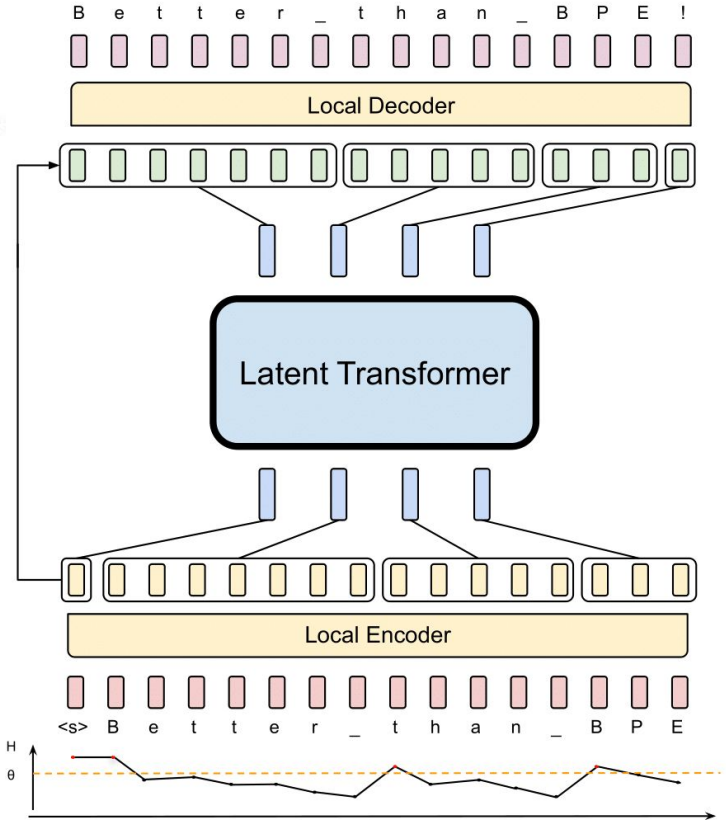
- Data driven approach
- Small auto-regressive LM to compute next byte entropies
- Splitting:
  - Threshold: H(x_t) > T
  - Difference Threshold: H(x_t) - H(x_{t-1}) > T

$$H(x_i) = \sum_{v \in \mathcal{V}} p_e(x_i = v|\boldsymbol{x}_{<i}) \log p_e(x_i = v|\boldsymbol{x}_{<i})$$



BLT Architecture





5. Small Byte-Level Transformer Makes **Next-Byte Prediction**

4. **Unpatching** to Byte Sequence via Cross-Attn

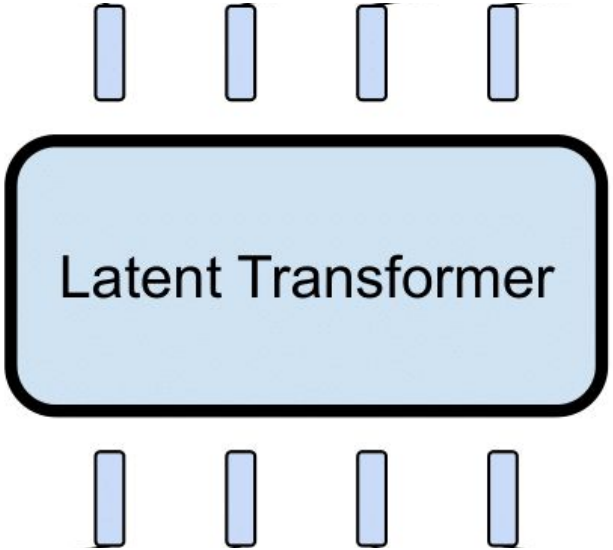
3. Large Latent Transformer **Predicts Next Patch**

2. Entropy-Based Grouping of Bytes Into **Patches** via Cross-Attn

1. Byte-Level Small Transformer Encodes **Byte Stream**

BLT: Latent Global Transformer

- Large autoregressive model
- Same as a regular LLM
- Maps sequence of patch inputs p_j to output representations o_j



BLT: Generating Local Encoder Embeddings from Bytes

- Start with individual byte embeddings (x_i).
- Extract byte n-grams (lengths 3 to 8) for context.
- Map n-grams using RollPolyHash (allows collisions).
- Sum the byte embedding (x_i) and its hashed n-gram embeddings.

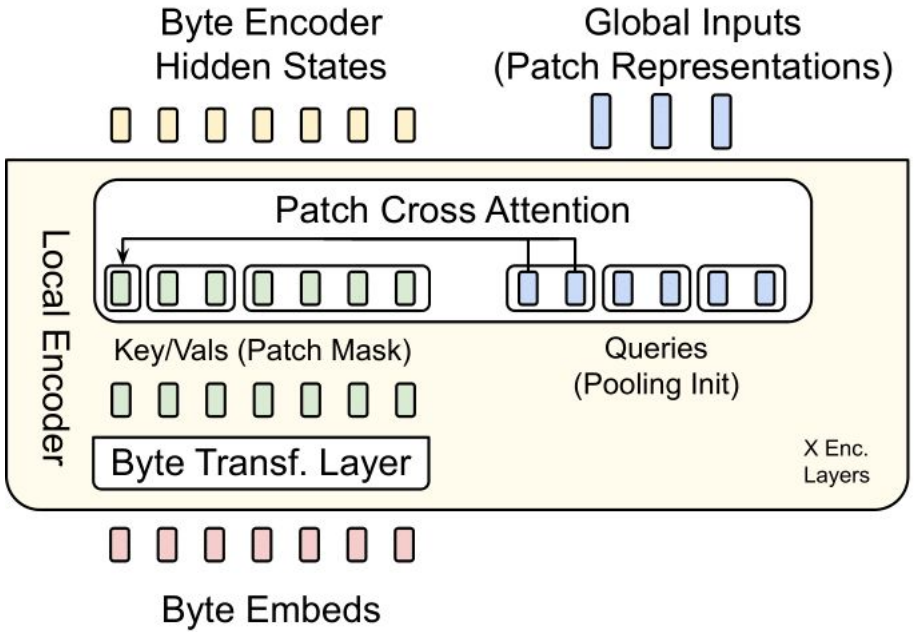
$$e_i = x_i + \sum_{n=3,\ldots,8} E_n^{hash}(\text{Hash}(g_{i,n}))$$

$$\text{where, } \text{Hash}(g_{i,n}) = \text{RollPolyHash}(g_{i,n}) \% |E_n^{hash}|$$

BLT: Local Encoder Model

Goal: Use byte embeddings add expressive information to patch embeddings using a small transformer.

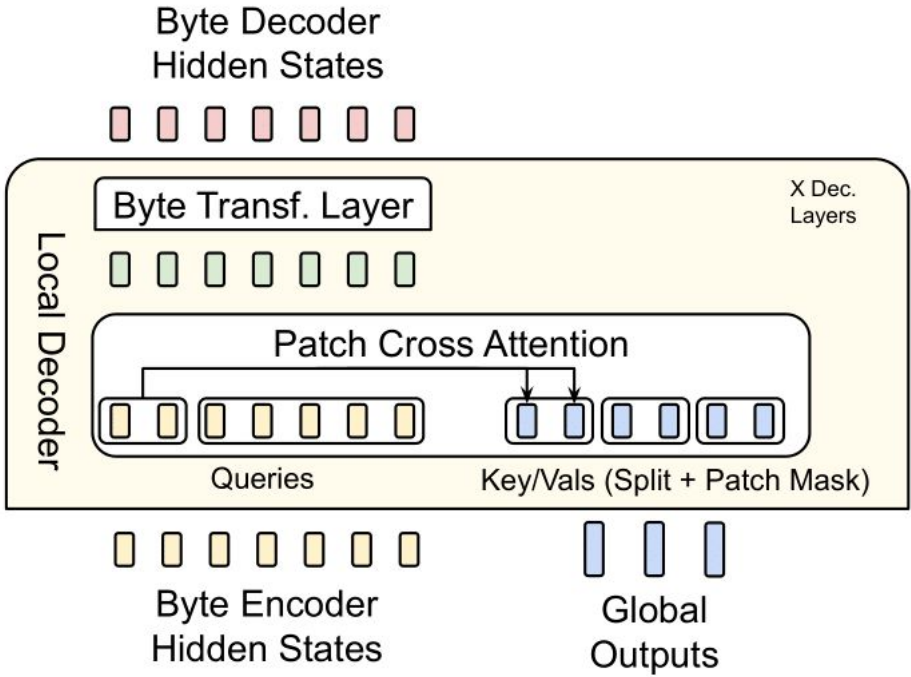
- Each patch starts with a query vector created by max pooling its bytes and a linear layer
- A small transformer produces contextual byte representations with local attention
- Patch queries attend only to the bytes inside their patch (patch mask).
- Alternating cross-attention and local attention layers adds local information to patch representation



BLT: Local Decoder

Goal: Use global patch representations after latent global transformer to update byte-level states from encoder.

- Patches now inform bytes instead of the other way around
- Cross-Attention: Each byte attends only to its patch's global representation
- Updates bytes local information with global context.
- Still updates the byte with local information after the cross attention



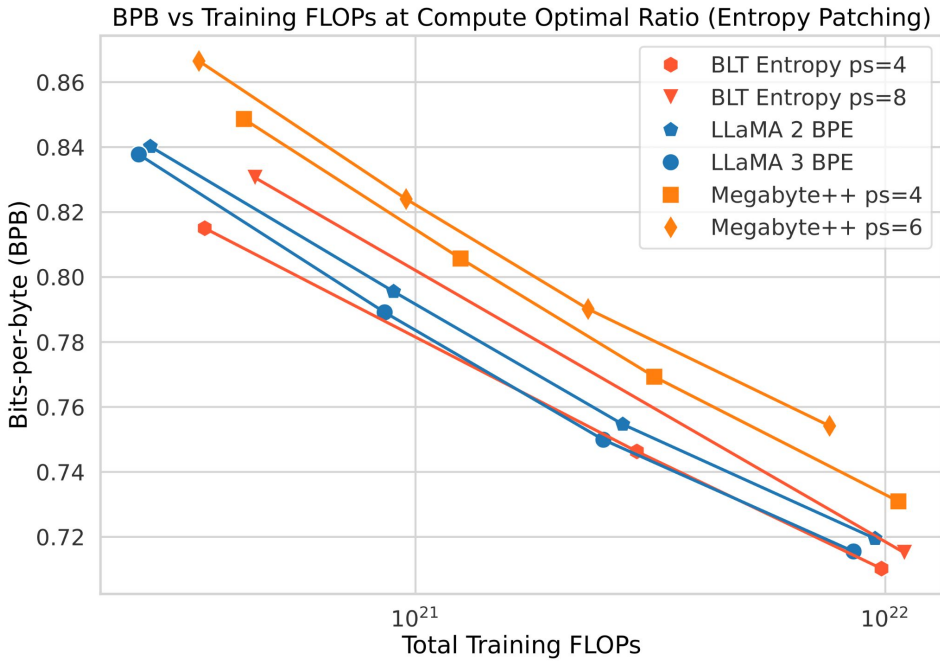
Experimental Setup

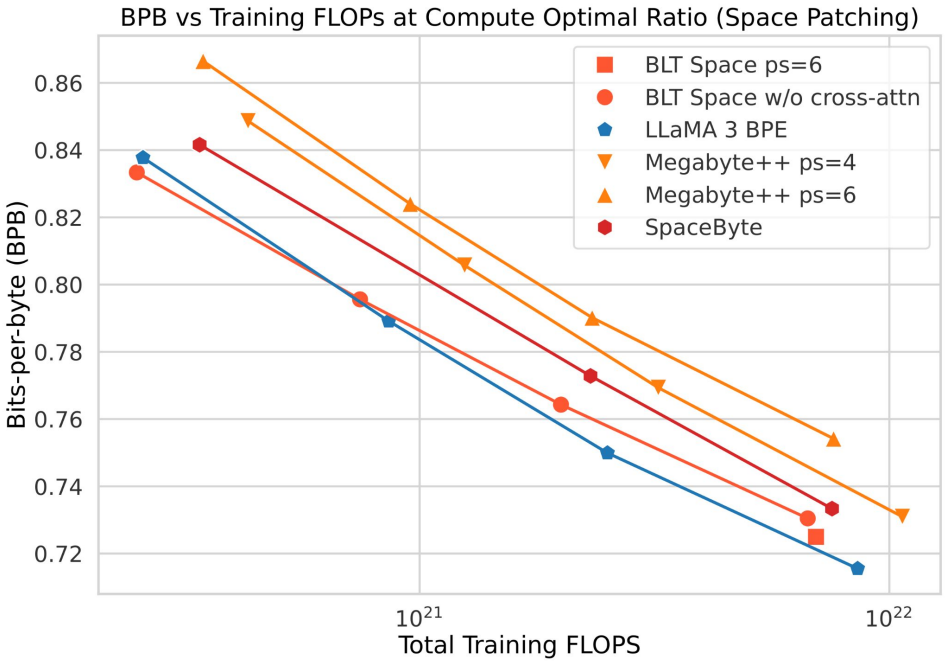
- Pre-training datasets
  - Llama 2 dataset (2 trillion tokens) - used for architectural choices/scaling laws
  - BLT 1T (public data + DataCompLM) - used to compare to LLaMa 3
- Entropy model
  - LM trained on same training distribution
  - 100M parameters with sliding window of 512 bytes
- Entropy Threshold: Choose threshold that achieves average patch size on pretraining mix

Scaling Trends

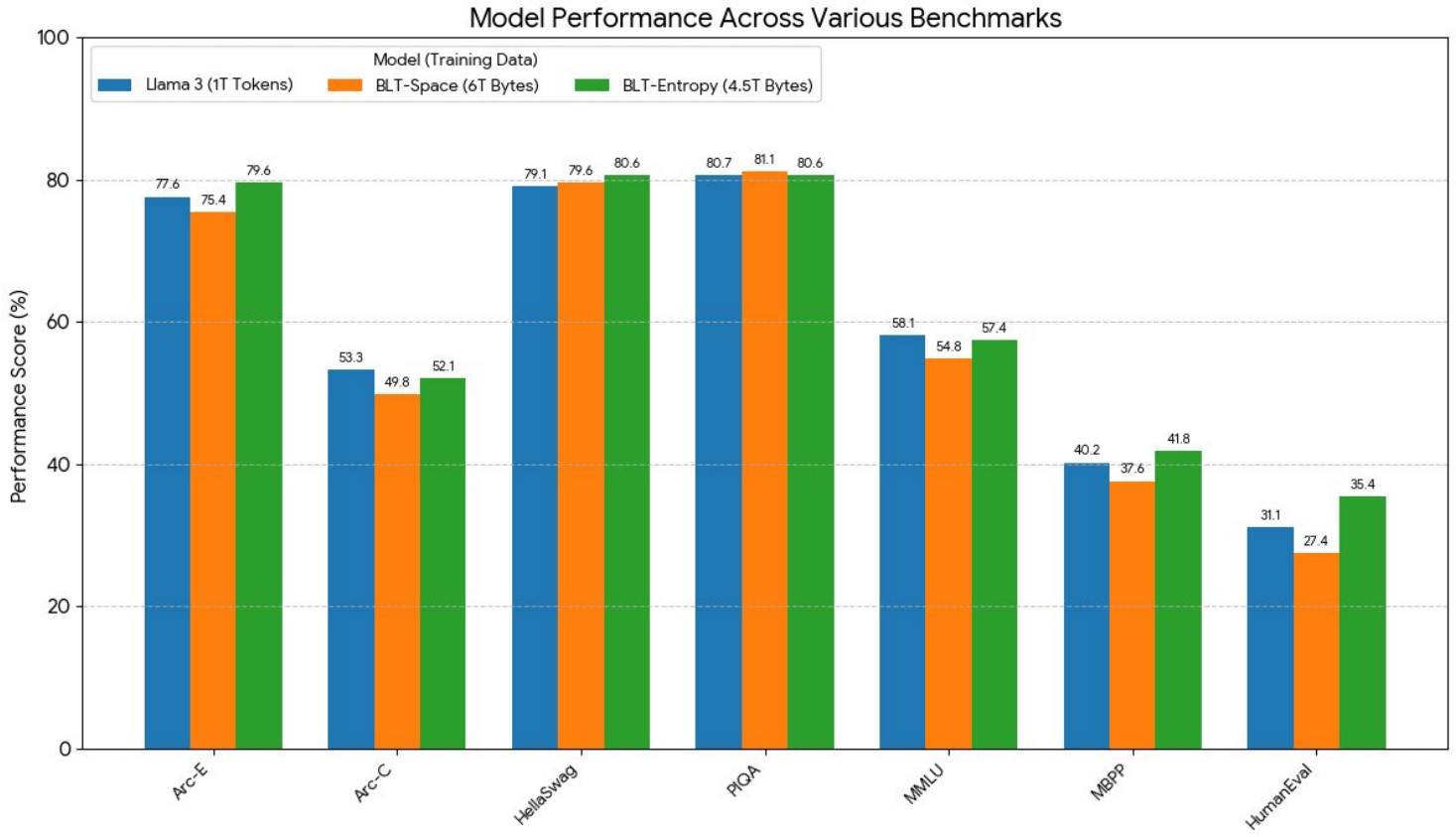
BPB = Tokenizer independent perplexity

$$\mathrm{BPB}(x) = \frac{\mathcal{L}_{CE}(\boldsymbol{x})}{\ln(2) \cdot n_{\text{bytes}}}$$

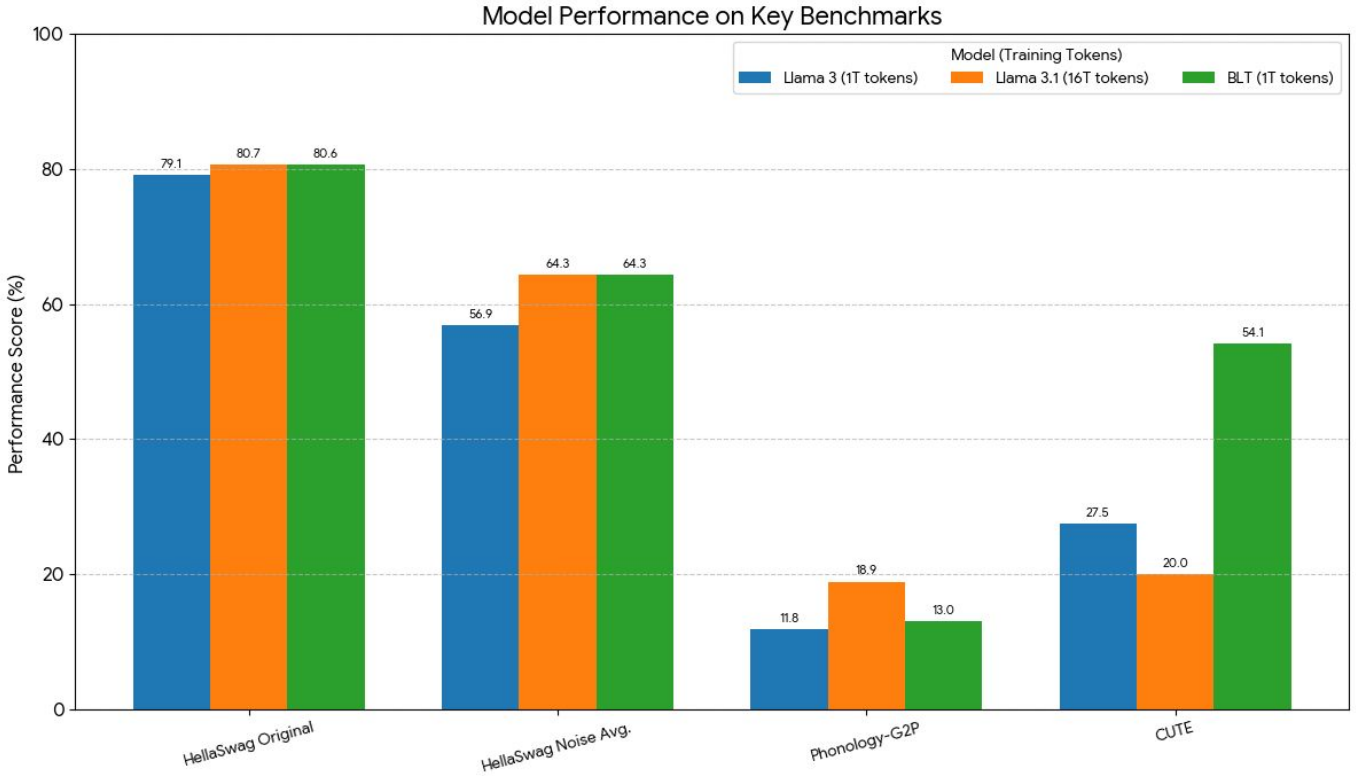




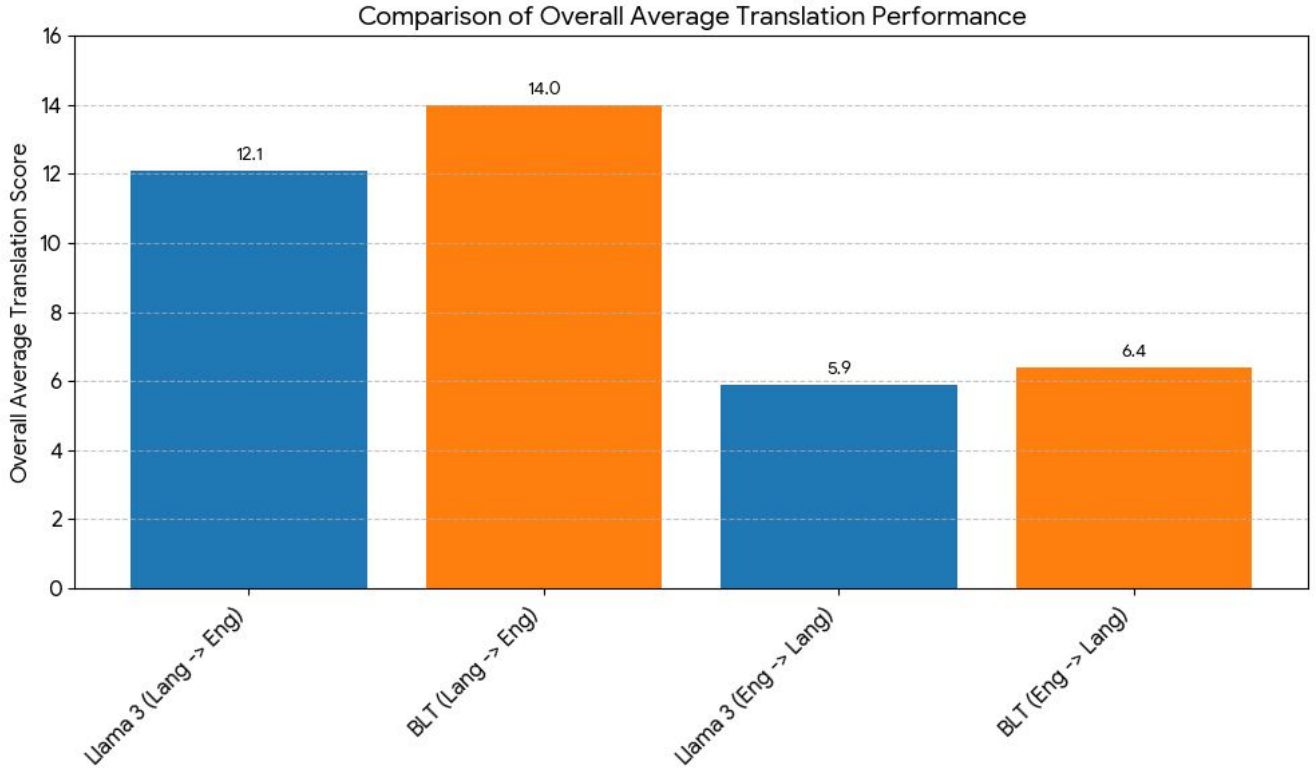
Performance



Improves Robustness



Low Resource Machine Translation



Limitations

- The entropy threshold
  - Passed the buck down to the human at inference time
  - Can we have a no hyperparameter method?
- The entropy patch model
  - Seems like the tokenizer bias lies in the model
  - Can we have this be learned end to end?
- The local encoder and decoder model architectures
  - Are these truly optimal?
  - How did they choose them?

The following are multiple choice questions (with answers) about college physics.

A refracting telescope consists of two converging lenses separated by 100 cm. The eye-piece lens has a focal length of 20 cm. The angular magnification of the telescope is
A. 4
B. 5
C. 6
D. 20
Answer: A
...
The muon decays with a characteristic lifetime of about 10^-6 second into an electron, a muon neutrino, and an electron antineutrino. The muon is forbidden from decaying into an electron and just a single neutrino by the law of conservation of
A. charge
B. mass
C. energy and momentum
D. lepton number
Answer: D
The quantum efficiency of a photon detector is 0.1. If 100 photons are sent into the detector, one after the other, the detector will detect photons
A. an average of 10 times, with an rms deviation of about 3
B. an average of 10 times, with an rms deviation of about 1
C. an average of 10 times, with an rms deviation of about 0.1
D. an average of 10 times, with an rms deviation of about 0.01
Answer: A

Questions

- Does this actually prove that tokenization is the root of all evil? Are we barking up the wrong tree in trying to keep text as the input?
- Could these insights of dynamic allocation be brought to other long context tasks or other modalities?
- Will these models suffer the same or worse fate as transformers in the long context regime?
- Will these models scale to the 10s of billions?

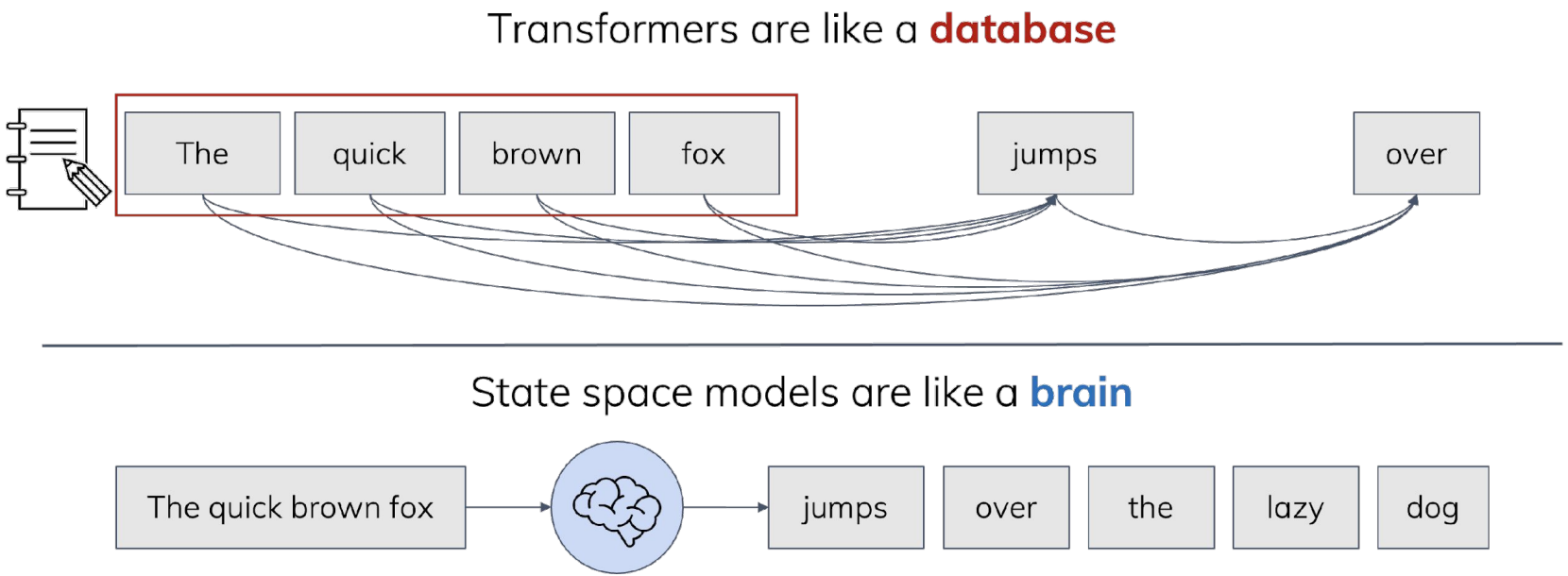
Dynamic Chunking for End-to-End Hierarchical Sequence Modeling

Sukjun Hwang
Carnegie Mellon University
sukjunh@cs.cmu.edu

Brandon Wang
Cartesia AI
brandon.wang@cartesia.ai

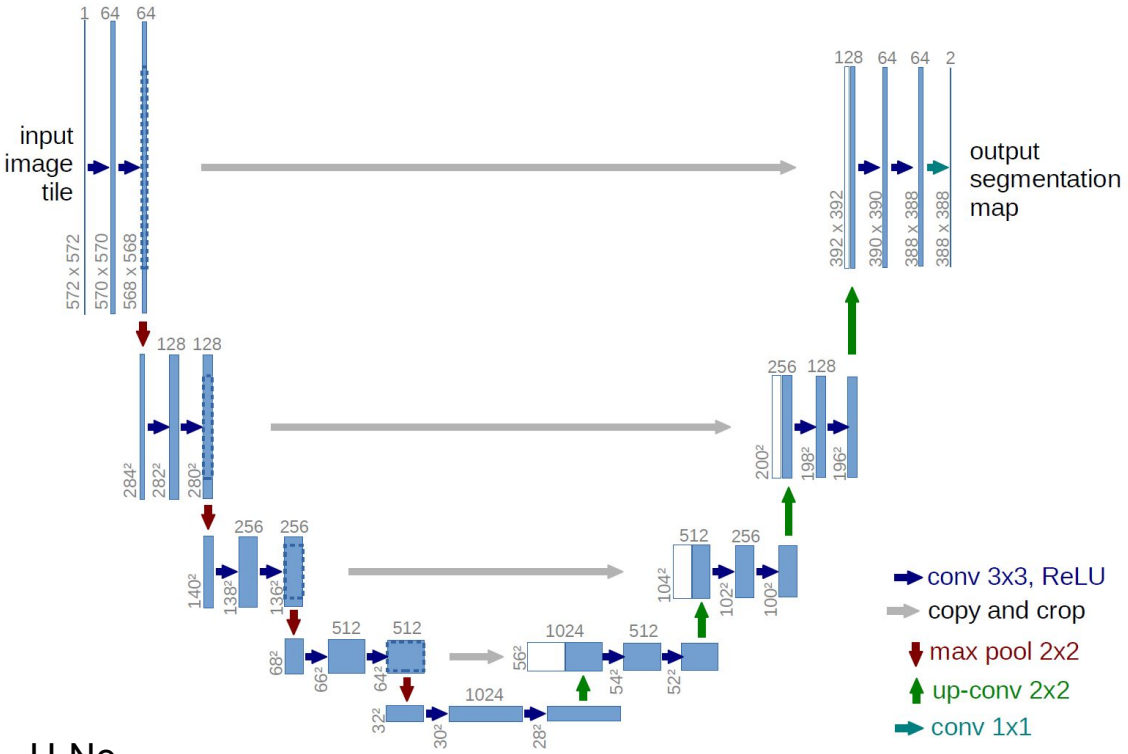
Albert Gu
Carnegie Mellon University, Cartesia AI
agu@cs.cmu.edu, albert@cartesia.ai

Aside: Mamba (SSM)

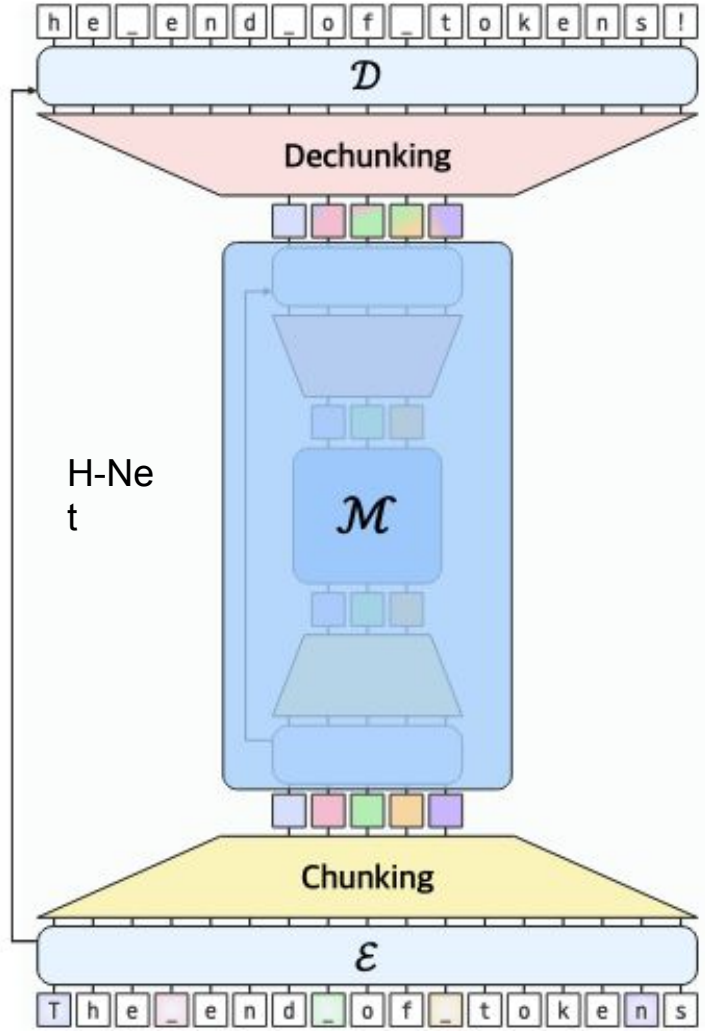


Intuitive blog by Mamba's Author (Albert Gu):
https://goombalab.github.io/blog/2025/tradeoffs/

H-Net: Hierarchical Processing



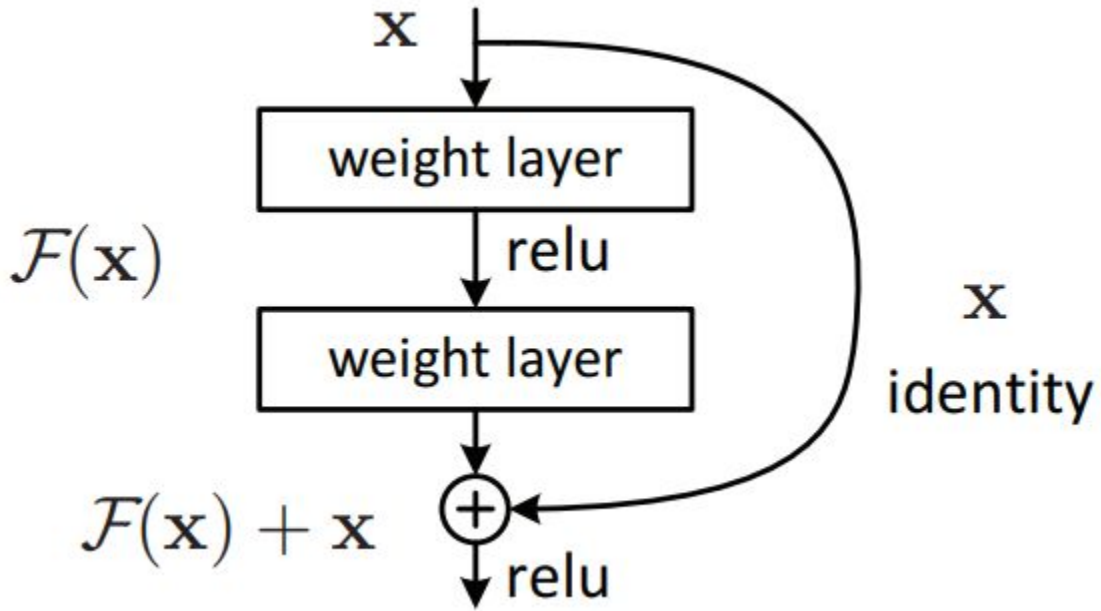
U-Net



H-Net

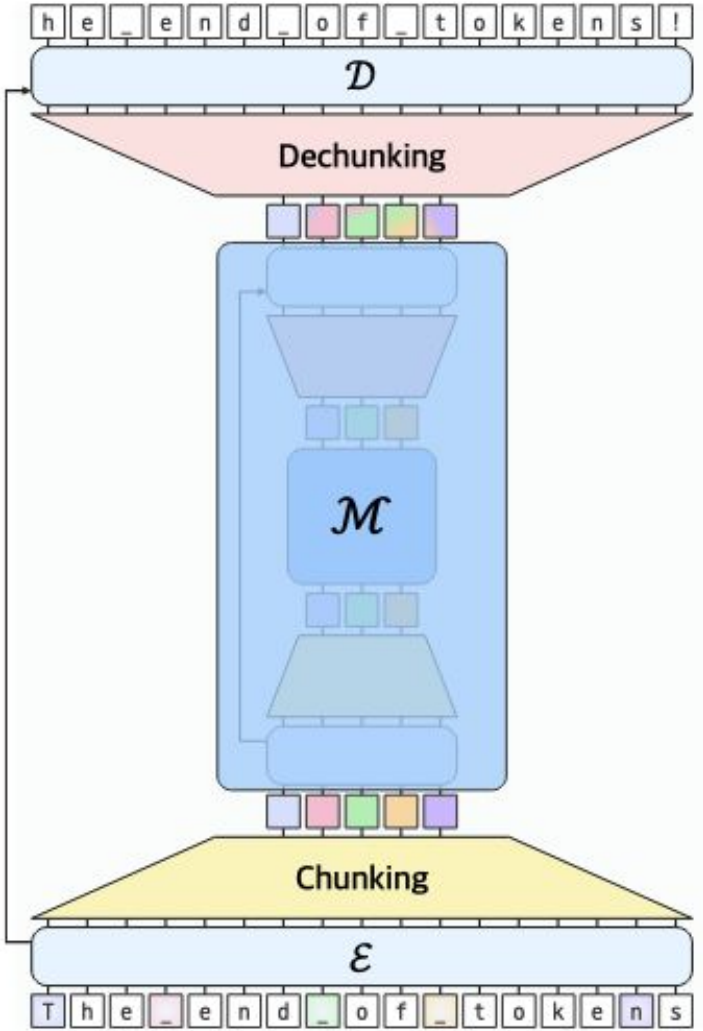
Residual Connections

- Popularized by ResNet in 2015
  - Understood as a solution to vanishing gradients
  - Optimization “highways”
- Ensemble of many shallow networks
- Feature reuse
  - Early layers’ features can be accessed by deeper layers



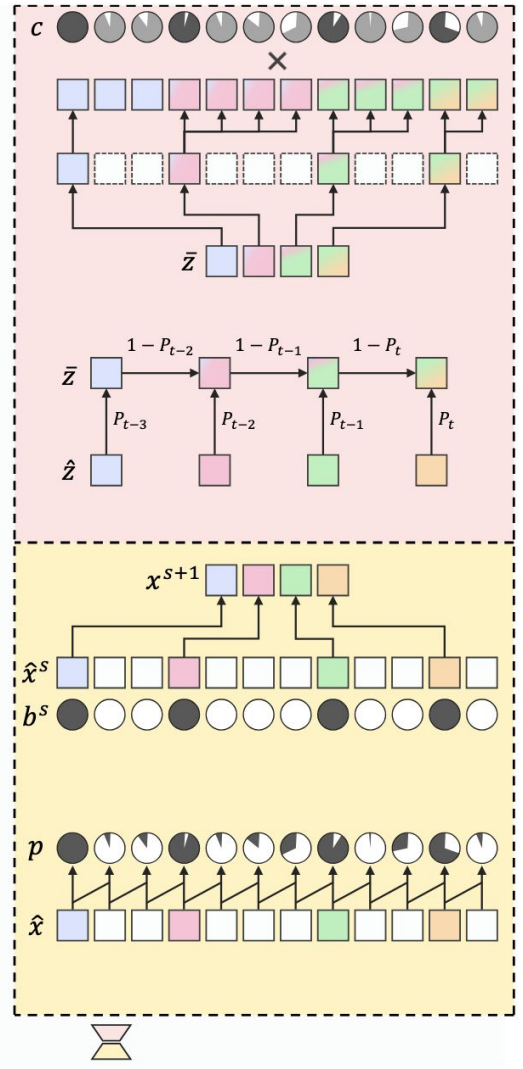
Architectural Overview

- H-Net progressively compresses
  - s = stage index (0 at byte level, S at innermost)
- Encode: E^s processes input x^s → output x̂^s
- Chunk: compress x̂^s →x^(s+1) + p^s
  - Shorter sequences + boundary probabilities
- Main network M processes x^S
  - ẑ^S = M (x^S)
- Dechunk: expand ẑ^s back to original resolution using p^s
- Decode: D^s produces final output



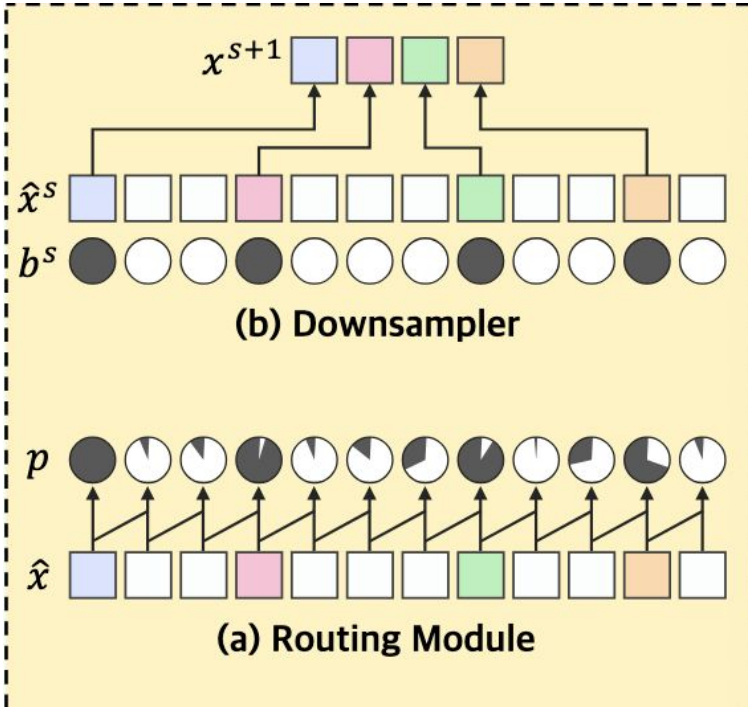
Dynamic Chunking (DC) Overview

- Routing module (bottom): predicts boundaries between adjacent elements through a similarity score
  - Chunk boundaries are drawn
  - Downsampler selectively retains semantically significant positions
- Dechunking
  - Smoothing module: discrete → interpolated representation
  - Upsampler



Chunking: Router Module + Downsampling

- Detects semantic shifts by measuring similarity between adjacent positions
  - High similarity between consecutive positions → low p_t (no boundary)
  - Low similarity between consecutive positions → high p_t (likely boundary)



Projections W_q and W_k are learned matrices that project encoder outputs into query/key spaces

Cosine similarity between **current** and **previous** positions

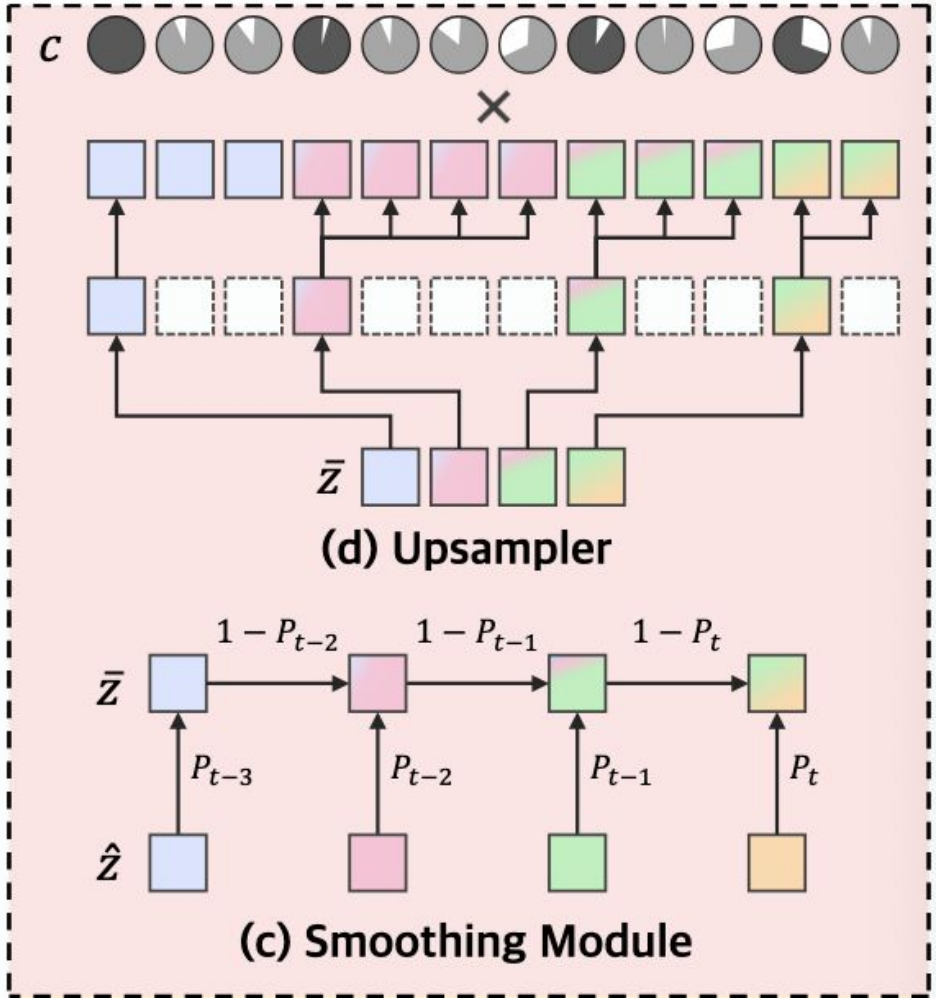
Direct selection: only keep vectors b_t = 1

$$q_t = W_q \hat{x}_t, \quad k_t = W_k \hat{x}_t, \qquad p_t = \frac{1}{2}\left(1 - \frac{q_t^\top k_{t-1}}{\|q_t\| \, \|k_{t-1}\|}\right) \in [0,1], \quad b_t = \mathbb{1}_{\{p_t \geq 0.5\}},$$

Dechunking: Smoothing

- Chunk boundaries are discrete by nature, which impedes gradient flow
- High confidence → use current chunk
- Low confidence → blend with previous using Exp. Moving Avg

$$\bar{z}_t = P_t \hat{z}_t + (1 - P_t)\bar{z}_{t-1}.$$



Dechunking: Upsampler

- Step 1: confidence scoring
  - Set c_t to p_t if b_t = 1
  - b_t = 0 → c_t = 1-p_t

$$c_t = p_t^{b_t}(1-p_t)^{1-b_t} = \begin{cases} p_t & \text{if } b_t = 1, \\ 1-p_t & \text{otherwise,} \end{cases}$$

- Step 2: gradient stabilization
  - ”Straight Through Estimator” (STE) rounds confidence to 1 in fwd pass
  - Backward pass preserves gradients: ∇STE(c_t) = ∇c_t

$$\mathrm{STE}(c_t) = c_t + \mathrm{stopgradient}(1-c_t),$$

$$\tilde{z}_t = \bar{z}_{\sum_{k=1}^{t} b_k},$$

- Step 3: causal expansion
  - Each position gets information from most recent chunk

Boundaries $\tilde{z}$: [1, 0, 0, 1, 0, 1]
Chunk indices $\bar{z}$: [1, 1, 1, 2, 2, 3]

- Step 4: Confidence-weighted output

$$\mathrm{Upsampler}(\bar{z}, c)_t = \mathrm{STE}\,(c_t) \cdot \tilde{z}_t.$$

Ratio Loss: how to prevent trivial chunking strategies

- Degenerate strategies are learned without guidance
- Ratio loss:
  - F represents fraction of positions selected as boundaries
  - G is average boundary probability
  - N is target compression ratio

$$\mathcal{L}_{\text{ratio}} = \frac{N}{N-1}\left((N-1)FG + (1-F)(1-G)\right), \qquad F = \frac{1}{L}\sum_{t=1}^{L} b_t, \quad G = \frac{1}{L}\sum_{t=1}^{L} p_t,$$

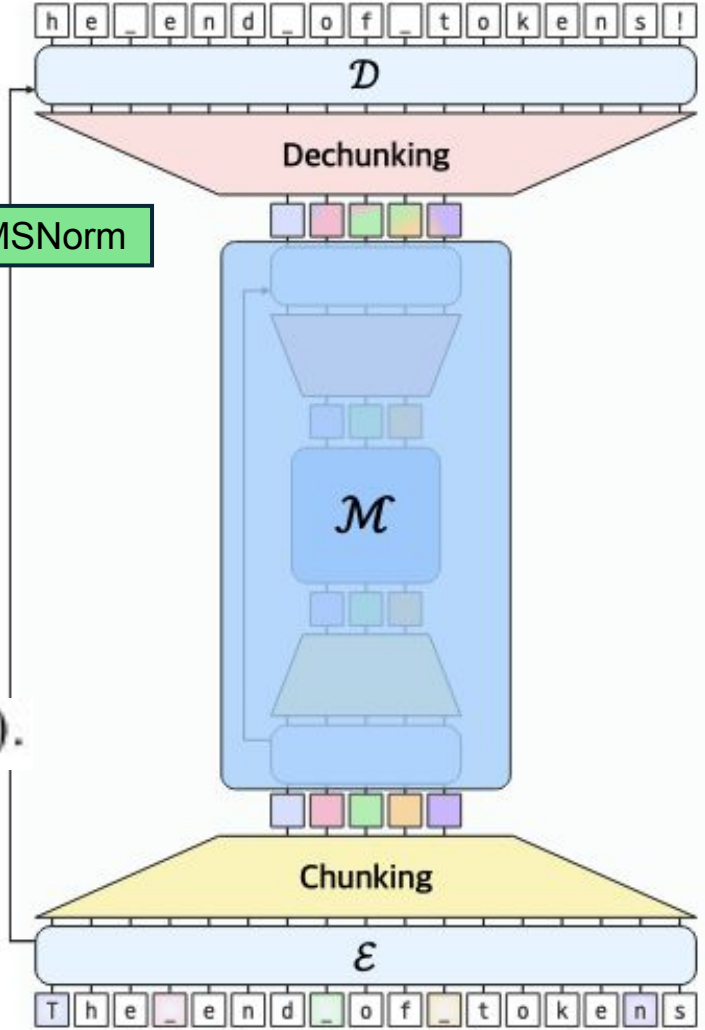
- Loss minimized when F = G = 1/N
  - Ratio Loss helps balanced utilization, creates pressure for more confident boundary decisions

Training Practices for Hierarchical Sequence Modeling

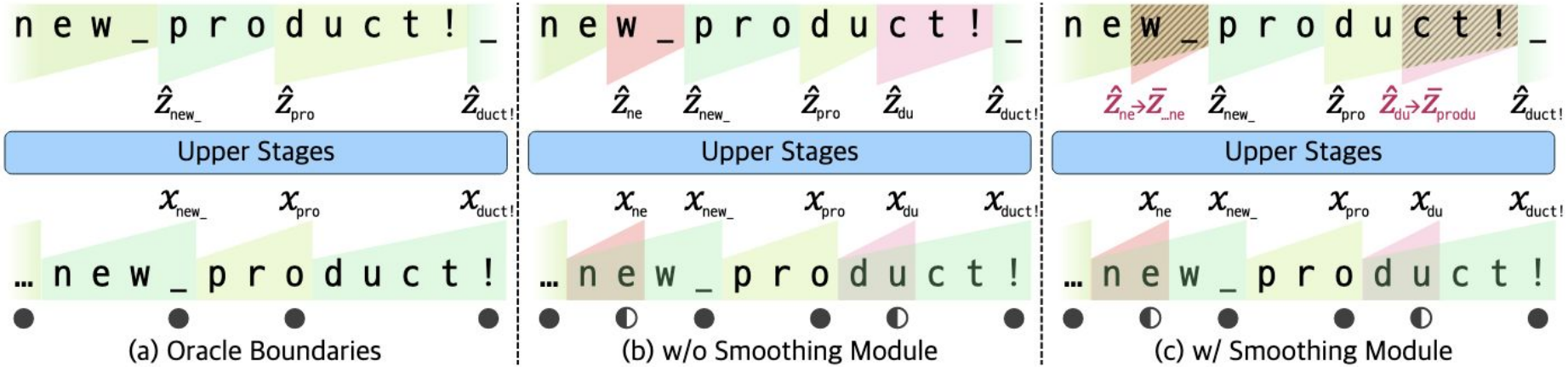
- Pre-normalization residual flow:
  - X → LayerNorm → Attention → X + F(X)
  - H-Net uses RMSNorm at the end of each network, so that contributions from each network are balanced
- Encoder has 2 roles
  1. Process with sequence-mixing layers to next stage
  2. Share features with decoder via residual connections

$$z^s = \text{Dechunk}(\hat{z}^{s+1}, p^s) + \text{Linear}(\hat{x}^s).$$

- Learning rate modulation:
  - Outer stages (longer sequences) get higher learning rates
  - outer stages directly influence the chunk boundaries that inner stages depend on

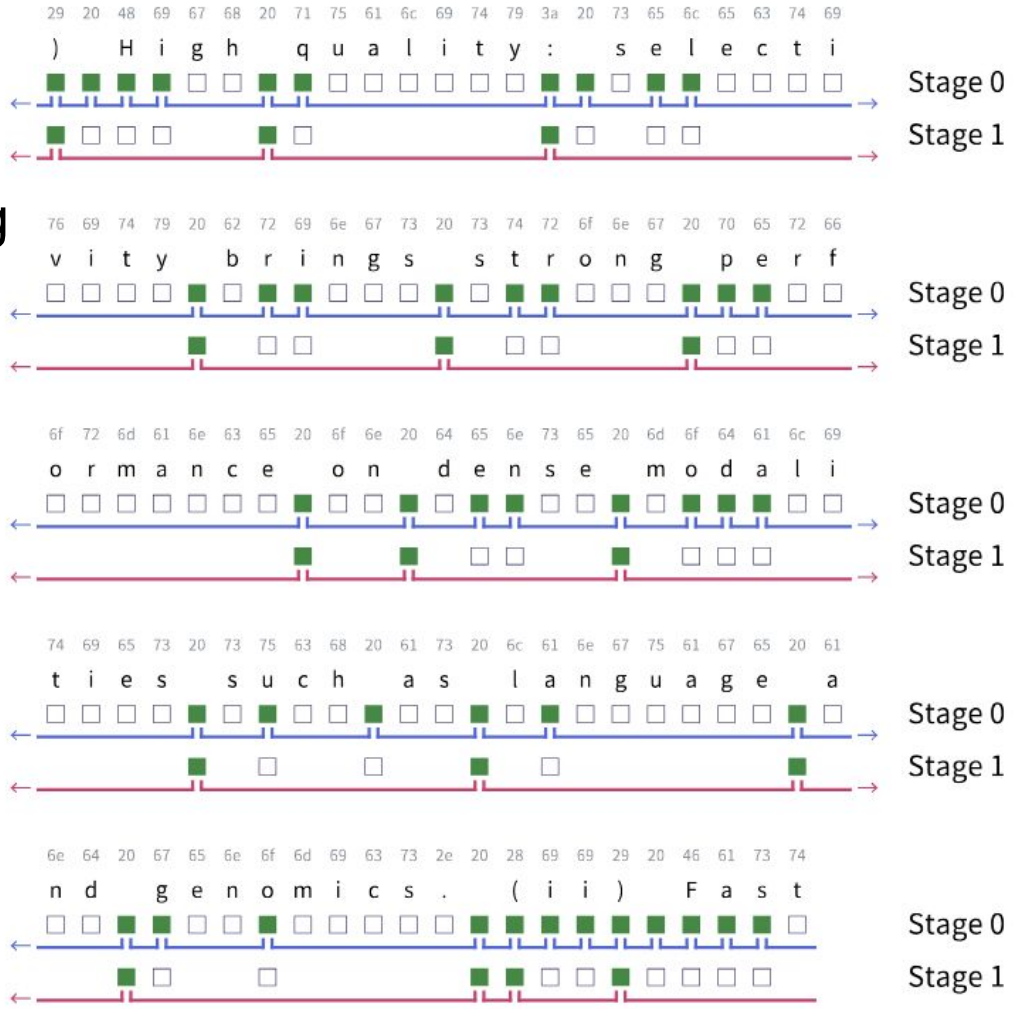


Chunking Example

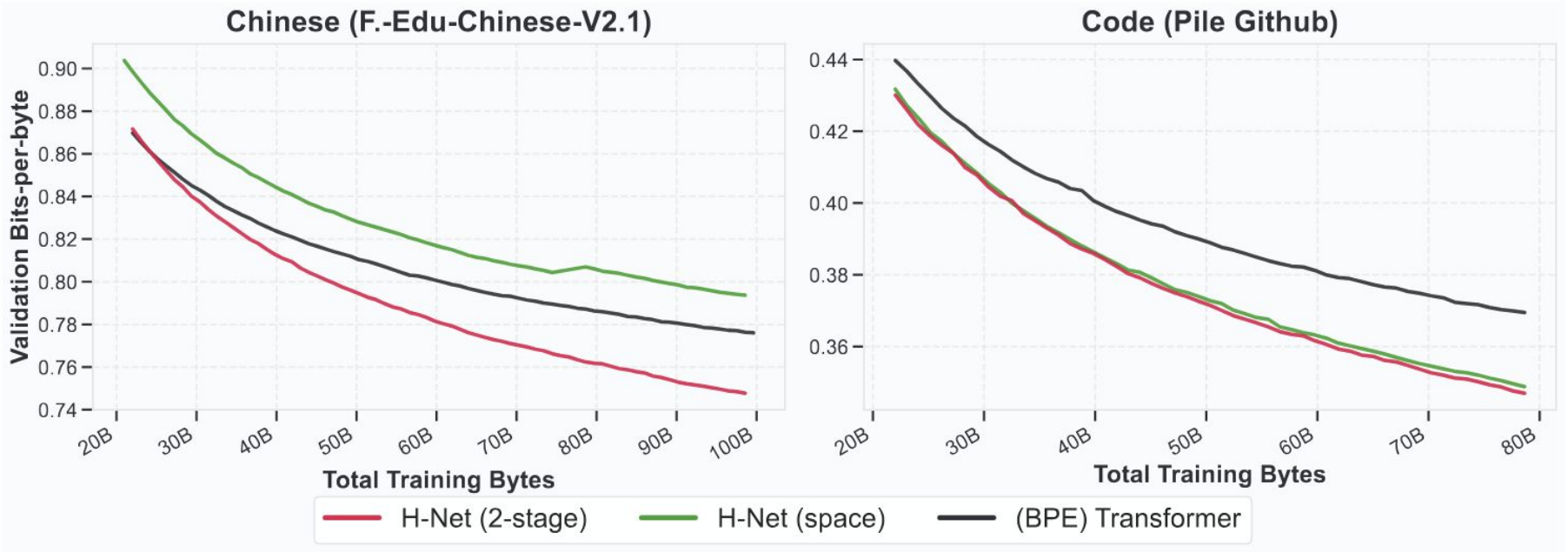


Recursion/Staging

- Raw bytes → Stage 0 chunking → Stage 1 chunking → Main

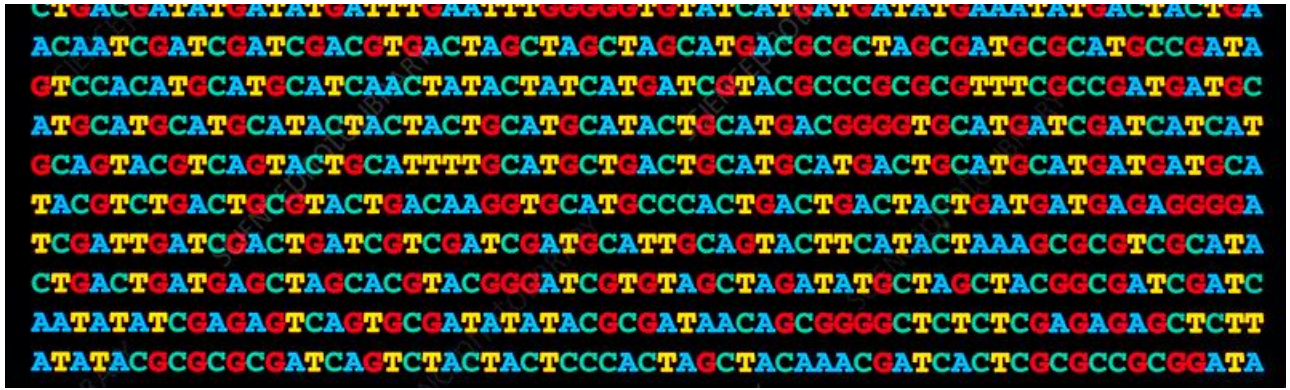


Weak Tokenization Heuristics



Human Genome Evaluation

- No natural segmentation cues (spaces, punctuation)
- Uniform, low complexity alphabet
- Context-dependent function
- Hierarchical structure: codons → genes → regulatory elements



Perplexity (ppl)

- "How many choices does the model think it has on average?"
  - Perplexity = exp(CE)
  - Lower ppl = better model
  - ppl of 4 = model is as confused as guessing among 4 options

Model / Architecture	Params.	Final ppl. ↓
Transformer (T9)	29M	2.769
Mamba-2 (M10)	33M	2.757
H-Net (M3T1 + T15 + M4)	64M	2.705
H-Net (M3T1 + M15 + M4)	66M	2.697

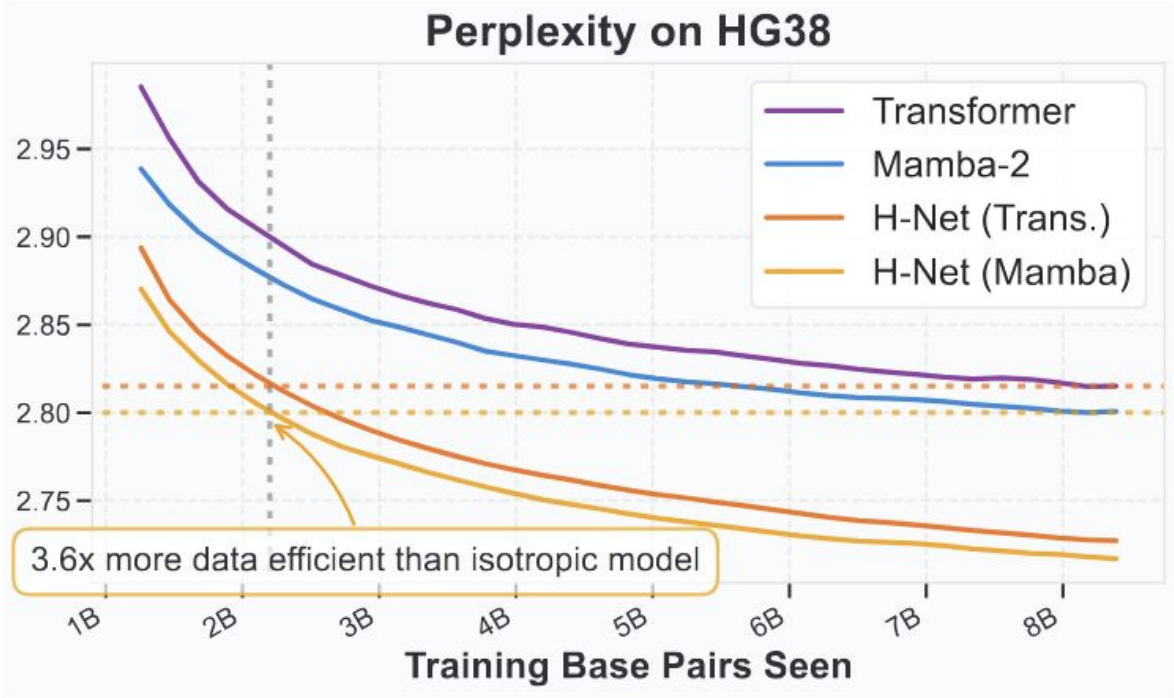
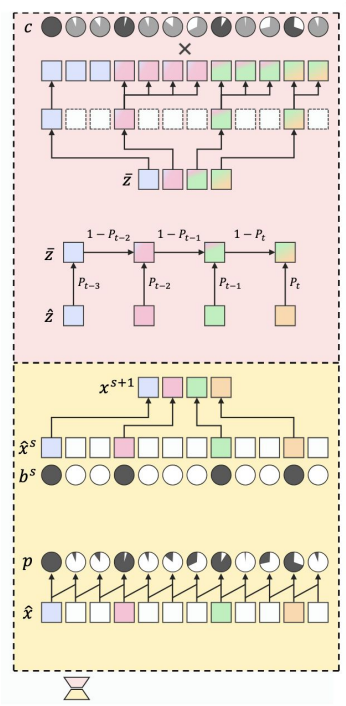


Figure 6: **Scaling performance on HG38** during the stable phase of training. Each H-Net model achieves the same pre-decay perplexity of the corresponding isotropic model with approximately 3.6× less data.

Main Evaluation

Table 1: **Architectures for main language models, all data-/FLOP-matched.** $\mathcal{E}^0, \mathcal{D}^0$, $\mathcal{E}^1, \mathcal{D}^1$, $\mathcal{M}$. T and M denote a Transformer and a Mamba-2 layer, respectively. For hierarchical byte-level models, the TOKENIZER column lists the chunking mechanism. The numbers before DC indicate downsampling factor N in equation (10); for example, (3,3)-DC denotes $N^0 = N^1 = 3$. The BPIC (Bytes-Per-Innermost-Chunk) measure shows that each chunk dynamically determined by our 1-stage comprises similar number of bytes to the GPT-2 tokenizer, despite aiming for $N^0 = 6$. All Transformer layers in $\mathcal{E}$ or $\mathcal{D}$ networks, as well as LlamaByte, use Sliding Window Attention (SWA) with a window size of 1024.

MODEL	INPUT	TOKENIZER	L^0	BPIC (L^S/L^0)	#PARAMS	ARCHITECTURE	d_model (D)
#FLOPs matched to GPT-3 *Large*							
Transformer	Token	GPT2	1792	4.6	760M	T24	1536
LlamaByte	Byte	—	8192	1.0	210M	T16	1024
MambaByte		—		1.0	190M	M28	1024
SpaceByte		Spacelike		6.0	570M	T8 + T16 + T8	768 , 1536
SpaceByte++		Spacelike		6.0	850M	M4 + T28 + M4	1024 , 1536
H-Net (pool)		6-Pool		6.0	850M	M4 + T28 + M4	1024 , 1536
H-Net (space)		Spacelike		6.0	850M	M4 + T28 + M4	1024 , 1536
H-Net (1-stage)		6-DC		4.8	680M	M4 + T22 + M4	1024 , 1536
H-Net (2-stage)		(3,3)-DC		7.0	870M	M4 + T1M4 + T26 + M4T1 + M4	1024 , 1024 , 1536
#FLOPs matched to GPT-3 *XL*							
Transformer	Token	GPT2	1792	4.6	1.3B	T24	2048
SpaceByte++	Byte	Spacelike	8192	6.0	1.6B	M4 + T31 + M4	1024 , 2048
H-Net (space)		Spacelike		6.0	1.6B	M4 + T31 + M4	1024 , 2048
H-Net (1-stage)		6-DC		4.7	1.3B	M4 + T24 + M4	1024 , 2048
H-Net (2-stage)		(3,3)-DC		6.9	1.6B	M4 + T1M4 + T27 + M4T1 + M4	1024 , 1536 , 2048



T: Transformer
M: Mamba (SSM)
BPIC: Compress. ratio

Main Evaluation

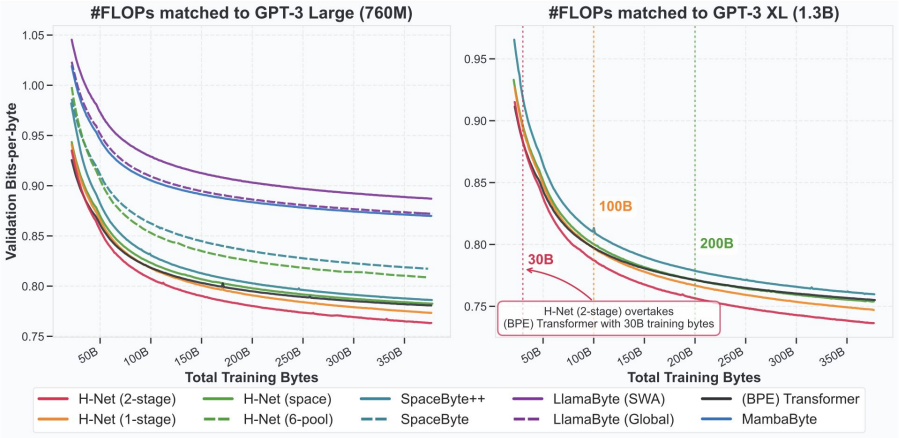


Figure 3: **Validation Bits-per-byte (BPB) throughout training** for different models at Large (760M, left) and XL (1.3B, right) scales with matched computational and data budgets for training. All models but Transformer take raw byte inputs (Transformer uses GPT-2 tokenizer). Vertical dotted lines indicate crossover points where H-Net begins to outperform Transformer with predefined BPE tokenization. From the curves we can clearly see the following: (1) all hierarchical models (*i.e.,* SpaceByte++, H-Net variants) outperform the isotropic models (*i.e.,* Transformer, MambaByte, LlamaByte); (2) dynamic chunking is more powerful than BPE tokenizers; and (3) DC is more effective than other chunking strategies. Furthermore, H-Net's 2-stage variant consistently outperforms 1-stage across both scales, demonstrating the effectiveness of deeper hierarchies. See Table 1 for architectural details.

Table 2: **Zero-shot performance comparison across multiple benchmarks, all data-/FLOP-matched.** Evaluation results on seven downstream tasks at both Large (760M) and XL (1.3B) scales. GFLOPs/Byte is measured during evaluation on FineWeb-Edu validation set. See Table 1 for architectural details.

Model	Input	GFLOPs/ Byte	F-Edu bpb ↓	LMB. acc ↑	Hella. acc_n ↑	PIQA acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	Wino. acc ↑	Open. acc_n ↑	Average acc ↑
#FLOPs matched to GPT-3 *Large*											
Transformer	Token	0.42	0.756	45.0	54.5	72.3	69.9	36.3	55.9	38.8	53.3
LlamaByte	Byte	0.42	0.859	37.0	40.5	64.7	55.1	26.7	52.3	32.4	44.1
LlamaByte (Global)		0.95	0.845	36.4	41.5	65.7	57.2	27.1	49.8	32.2	44.3
MambaByte		0.42	0.845	32.9	42.0	66.2	55.9	28.1	51.7	33.2	44.3
SpaceByte		0.41	0.791	43.0	49.0	69.0	63.3	33.5	53.3	35.0	49.4
SpaceByte++		0.42	0.760	**48.0**	55.7	71.3	67.9	35.4	57.5	39.6	53.6
H-Net (pool)		0.42	0.780	43.2	54.7	69.7	67.9	34.7	54.8	36.4	51.6
H-Net (space)		0.42	0.755	46.7	55.9	**72.4**	68.8	34.6	57.6	38.0	53.4
H-Net (1-stage)		0.43	0.755	46.2	55.5	71.0	68.1	35.6	58.6	40.0	53.6
H-Net (2-stage)		0.43	**0.743**	46.9	**57.4**	72.0	**71.7**	**39.2**	**60.4**	**40.6**	**55.5**
#FLOPs matched to GPT-3 *XL*											
Transformer	Token	0.69	0.730	48.1	58.0	73.1	72.2	37.5	58.6	40.8	55.5
SpaceByte++	Byte	0.72	0.733	**51.3**	60.1	72.4	71.8	38.0	58.5	40.6	56.1
H-Net (space)		0.70	0.726	50.3	61.5	73.6	72.4	40.2	60.2	41.8	57.1
H-Net (1-stage)		0.72	0.728	48.4	59.5	72.4	73.0	38.3	59.2	42.4	56.2
H-Net (2-stage)		0.69	**0.715**	50.5	**62.2**	**73.7**	**74.2**	**42.2**	**60.5**	**44.0**	**58.2**

Robustness to Char-Level Noise

Model	Input	HellaSwag					Average ↑	Robustness Score ↑
		AntSpeak	Drop	RandomCase	Repeat	UpperCase		
#FLOPs matched to GPT-3 *Large*								
Transformer	Token	31.1	29.9	27.1	27.8	38.9	30.9	20.2
LlamaByte (W1024)	Byte	30.4	28.1	29.3	27.2	38.5	30.7	36.9
LlamaByte (Global)		31.1	28.1	29.7	27.3	39.0	31.0	36.6
MambaByte		29.8	27.9	29.9	27.1	39.6	30.9	34.5
SpaceByte		30.7	29.8	33.5	29.5	47.8	34.3	38.1
SpaceByte++		31.0	30.9	35.8	29.3	54.0	36.2	36.4
H-Net (pool)		30.5	31.2	35.4	29.6	53.4	36.1	37.3
H-Net (space)		30.8	31.2	38.6	29.4	54.0	36.8	38.2
H-Net (1-stage)		**31.2**	31.1	35.4	29.9	54.1	36.4	37.2
H-Net (2-stage)		30.8	**32.1**	**39.3**	**30.4**	**57.1**	**38.0**	**39.0**
#FLOPs matched to GPT-3 *XL*								
Transformer	Token	**31.6**	30.7	28.0	28.5	43.0	32.3	22.2
SpaceByte++	Byte	30.9	32.1	40.3	30.6	58.5	38.5	38.5
H-Net (space)		31.2	33.2	41.9	31.8	60.7	39.8	40.5
H-Net (1-stage)		30.9	32.7	39.2	31.2	58.4	38.6	39.5
H-Net (2-stage)		31.1	**34.7**	**44.1**	**33.0**	**61.7**	**40.9**	**42.8**

AntSpeak: uppercase, space-sep chars
Drop: remove 10% chars
RandomCase: convert cases randomly
Repeat: repeat 20% chars upto 4x
UpperCase: all caps

Ablation Study

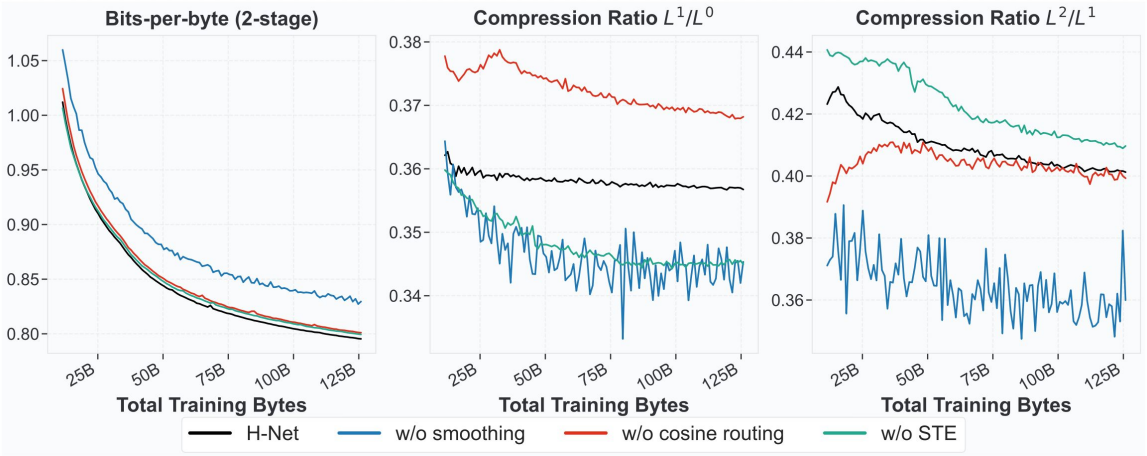
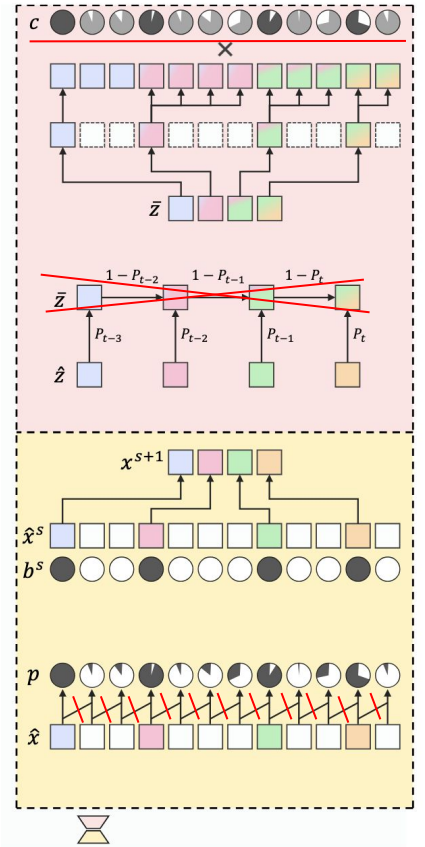


Figure 7: **Ablation study on key H-Net components** showing validation BPB (left) and compression ratios for the first stage L^1/L^0 (center) and second stage L^2/L^1 (right) during training. Using H-Net (2-stage), we evaluate the impact of removing three components: the smoothing module (w/o smoothing), the similarity-based routing module (w/o cosine routing), and Straight-Through Estimator (w/o STE).

STE makes each of these 1
no matter what c is



Ablation Study

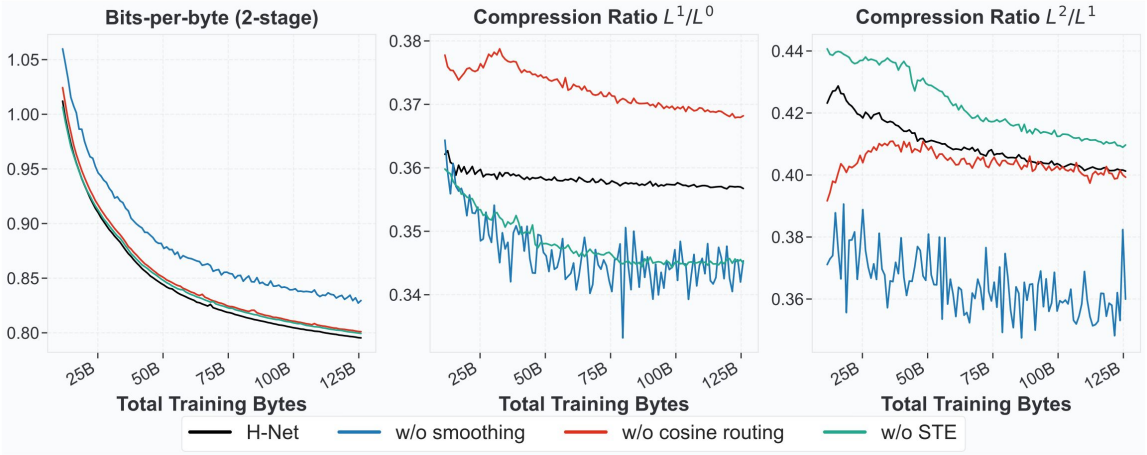


Figure 7: **Ablation study on key H-Net components** showing validation BPB (left) and compression ratios for the first stage L^1/L^0 (center) and second stage L^2/L^1 (right) during training. Using H-Net (2-stage), we evaluate the impact of removing three components: the smoothing module (w/o smoothing), the similarity-based routing module (w/o cosine routing), and Straight-Through Estimator (w/o STE).

Upsampler. We carefully design the upsampler (see Figure 1-(d)) that decompresses $\bar{z}^{s+1}$ to match the original resolution of inputs in the previous stage z^s with the following definition:

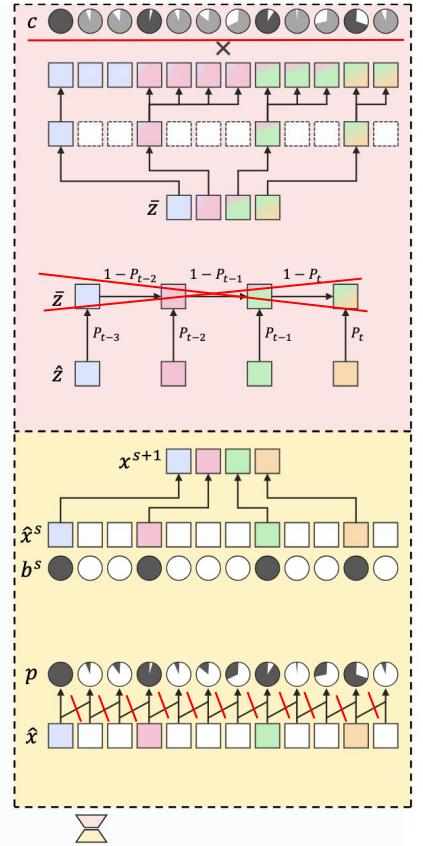
$$c_t = p_t^{b_t}(1-p_t)^{1-b_t} = \begin{cases} p_t & \text{if } b_t = 1, \\ 1-p_t & \text{otherwise,} \end{cases} \tag{6}$$

$$\text{STE}(c_t) = c_t + \text{stopgradient}(1-c_t), \tag{7}$$

$$\tilde{z}_t = \bar{z}_{\sum_{k=1}^{t} b_k}, \tag{8}$$

$$\text{Upsampler}(\bar{z}, c)_t = \text{STE}(c_t) \cdot \tilde{z}_t. \tag{9}$$

STE makes each of these 1
no matter what c is



Ablation Study – Encoder-Decoder Arch

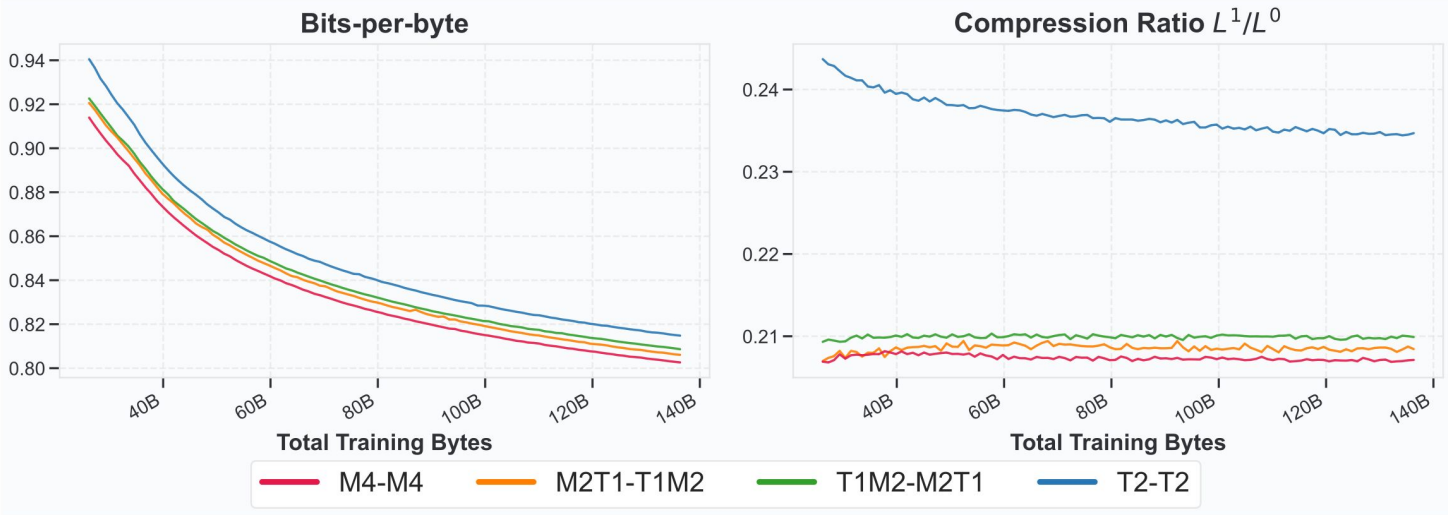


Figure 8: **Encoder-decoder architecture ablation using raw byte inputs.** Validation BPB (left) and compression ratio L^1/L^0 (right) for H-Net (1-stage) throughout training. We evaluate four encoder-decoder ($\mathcal{E}^0 - \mathcal{D}^0$) configurations: M4-M4 , M2T1-T1M2 and T1M2-M2T1, and T2-T2, where M denotes Mamba layers and T denotes Transformer layers.

Ablation Study – Encoder-Decoder Arch

From p.19
Q: Is Mamba better because of the fine-grained byte inputs? (Fact: Mamba > Trans on bytes)

-> use BPE as stage-1 and test stage-2 arch

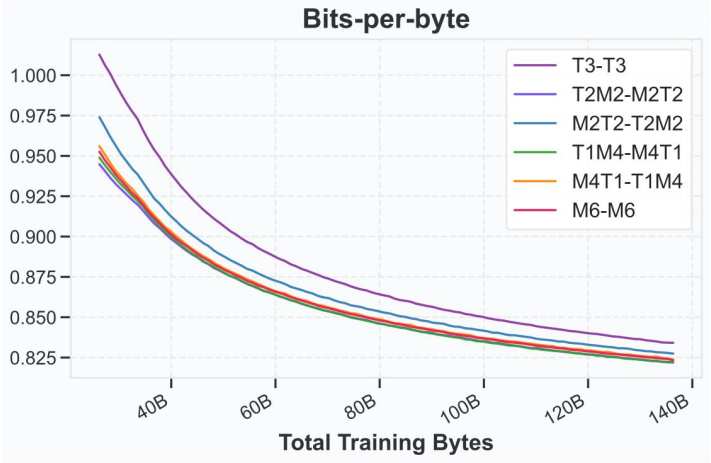


Figure 10: **Encoder-decoder architecture ablation using BPE-tokenized inputs.** Assuming that GPT-2 tokenizer serves as the outermost encoder-decoder (*i.e.*, $\mathcal{E}^0$-$\mathcal{D}^0$), we evaluate six $\mathcal{E}^1$-$\mathcal{D}^1$ combinations: M6-M6, M4T1-T1M4, T1M4-M4T1, M2T2-T2M2, T2M2-M2T2, and T3-T3.

Ablation Study – Main Net Arch

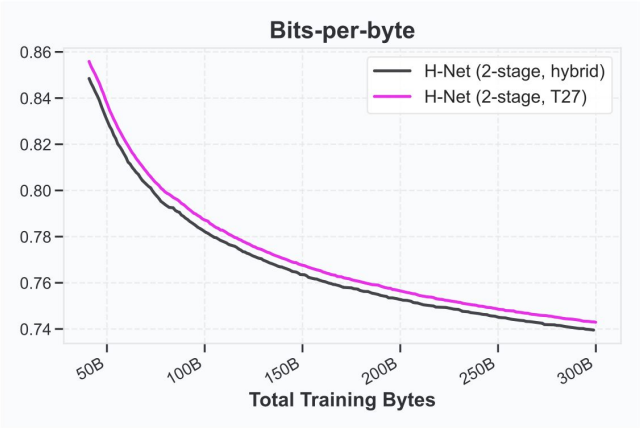


Figure 11: **Hybrid main network.** Bits-per-byte during the stable phase of training, for H-Net (2-stage) with Transformer main stage and with hybrid main stage. The hybrid main stage scales better, similar to findings for standard token-based language models. This finding suggests that design principles for isotropic (tokenized) models can carry over to choices of the main network.

Hybrid:
20 Mamba-2 and
7 Transformer layers
interleaved in a 3:1 ratio.

Compare to MoE (Sparse Network)

Same FLOPs but more params?
-> **UNFAIR!!**

H-Net is by-design sparse.
-> Compare it with MoE isotropic models

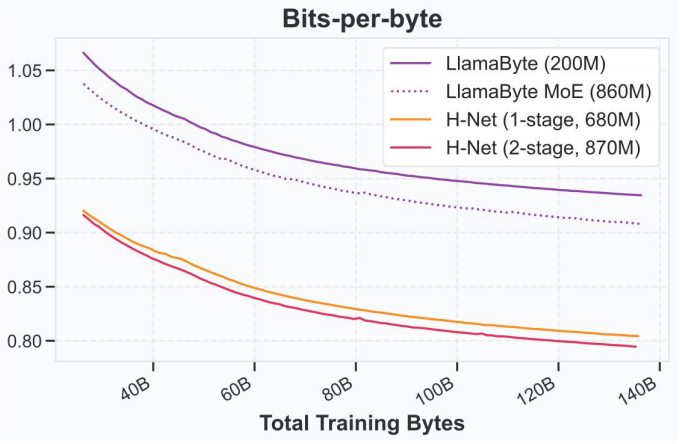
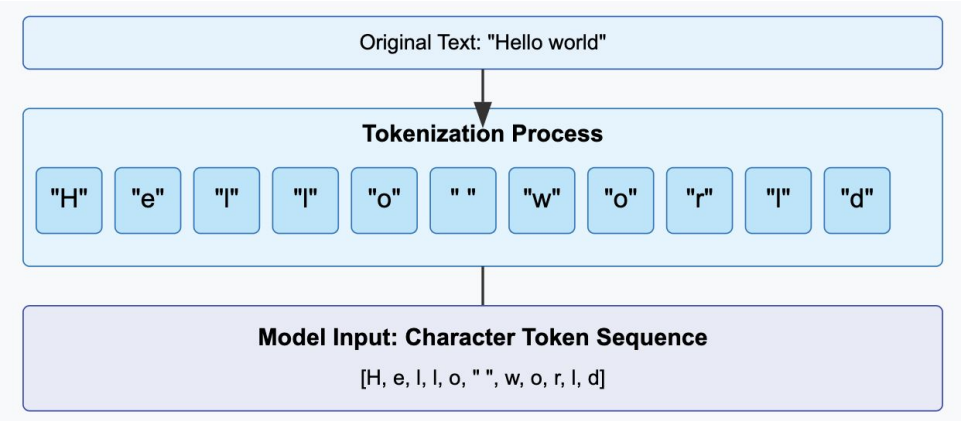


Figure 12: **Comparison to Mixtures-of-Experts.** Bits-per-byte comparison of H-Net (both 1-stage and 2-stage) to LlamaByte-MoE, which is a FLOPs-matched MoE model that uses a similar number of parameters as H-Net (2-stage). Both H-Nets perform much better than LlamaByte-MoE, implying that H-Net's capabilities do not just come from sparsity.

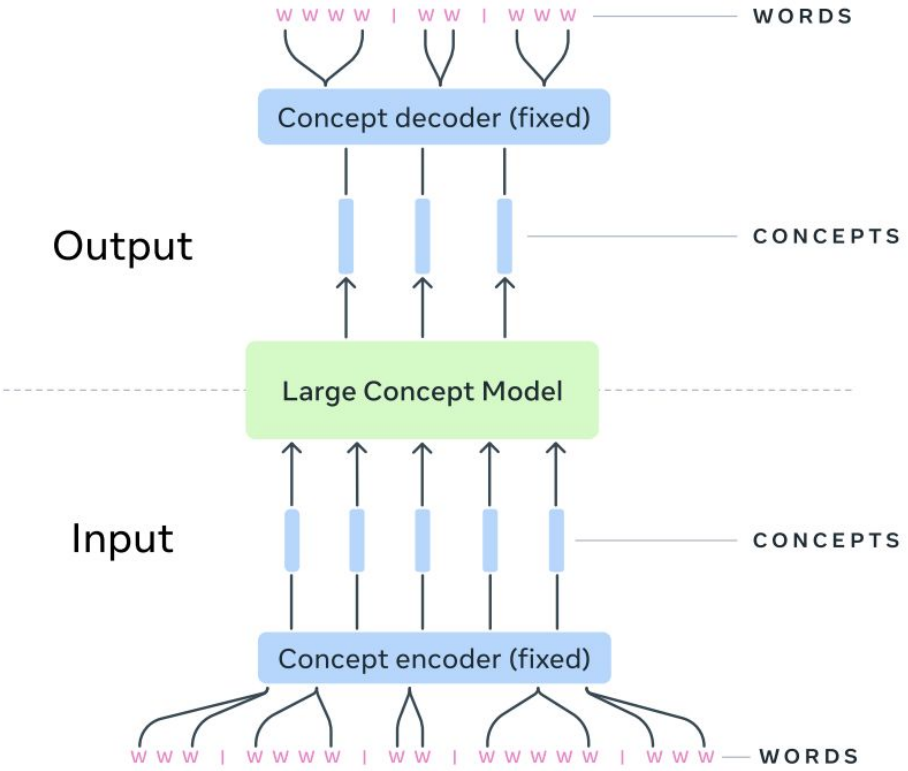
BLT Questions (again)

- Does this actually prove that tokenization is the root of all evil? Are we barking up the wrong tree in trying to keep text as the input?
- Could these insights of dynamic allocation be brought to other long context tasks or other modalities?
- Will these models suffer the same or worse fate as transformers in the long context regime?
- Will these models scale to the 10s of billions?

Where On the Tokenization Spectrum Should We Sit?

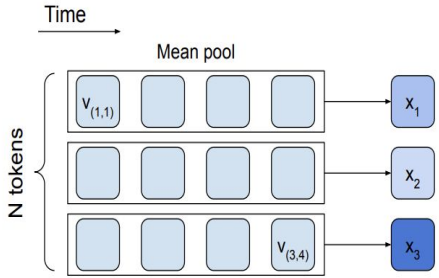


Character LMs

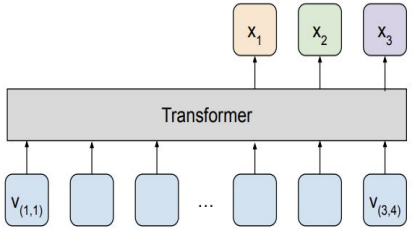


Concept LMs

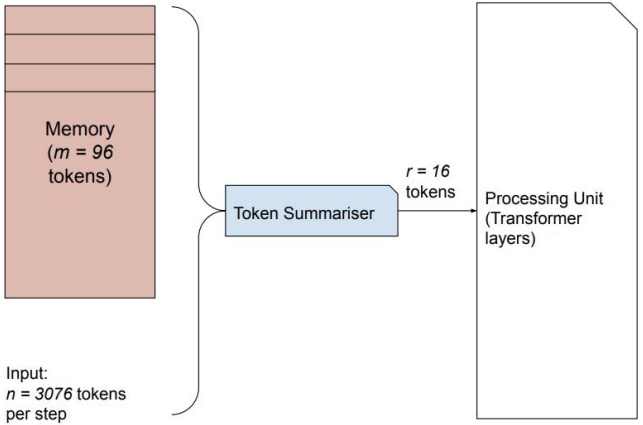
How much should we spend to create LM inputs?

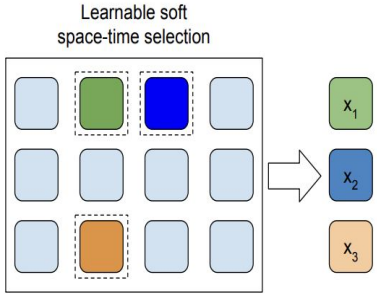


(a) Temporal pooling

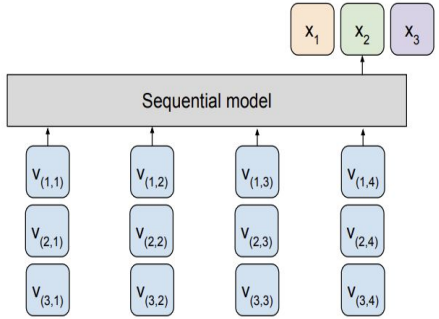


(b) Transformer-based





(c) Attentional pooling (TokenLearner)



(d) Sequential model (TTM)