LZ78 Tokenization Ablation Study — Experiment Report

Date: February 18, 2026 **Project**: nanochat **Authors**: Parsa Idehpour

---

1. Overview

This report documents a systematic ablation study comparing **LZ78-family tokenizers** against **BPE (Byte-Pair Encoding)** for language model training. We evaluate how tokenizer type, token embedding strategy, and loss function affect model quality, measured in **bits-per-byte (BPB)** — a vocab-size-independent metric enabling fair comparison across tokenizers with different vocabulary sizes.

Key research questions: 1. Can LZ78-based tokenizers match BPE performance for LM training? 2. Does exploiting the tree structure of LZ78 tokens (via structured embeddings) help? 3. Can prefix-aware soft labels improve training by giving partial credit to prefix tokens?

---

2. Experimental Setup

2.1 Model Architecture

All experiments use the same GPT model (based on nanochat):

Parameter	Value
Depth (n_layer)	12
Model dim (n_embd)	768
Attention heads	6 (head_dim = 128)
MLP	4x expansion, ReLU^2 activation
Positional encoding	Rotary (RoPE)
Normalization	RMSNorm (no learnable params)
Embedding / LM head	Untied weights
Context length	2048 tokens
Batch size	524,288 tokens
Optimizer	Muon (matrix layers) + AdamW (embeddings)
Training horizon	Chinchilla-20 (20x params in tokens)
Target steps	~5,133 (varies slightly by vocab size)
Precision	bfloat16

Parameter count varies by vocabulary size: - 32K vocab: ~134M params - 44K vocab: ~153M params

2.2 Dataset

- **Training data**: C4 (Colossal Clean Crawled Corpus)
- **Format**: 32 parquet shards (~2.9 GB total)
- **LZ78 tokenizers**: Data pre-tokenized into `.npy` shards (~1.1 GB each)
- **BPE tokenizer**: Tokenizes on-the-fly from parquet files
- **Evaluation**: Separate C4 validation split, 20 x 524,288 tokens

2.3 Evaluation Metric

Bits-per-byte (BPB): Converts the cross-entropy loss to bits per raw byte of text, enabling fair comparison across tokenizers with different vocabulary sizes and compression ratios. Lower is better.

---

3. Tokenizers

We compare four tokenizer families, all operating on raw bytes:

3.1 BPE (Byte-Pair Encoding) — Baseline

- **Vocab size**: 32,768
- **Method**: Standard BPE trained on C4 via RustBPE
- **Properties**: Iteratively merges the most frequent byte pair. No tree structure.
- **Tokenizer path**: `/large_storage/.../nanochat/tokenizer-32k/`
- **Training time**: 243 seconds

3.2 LZ78 (Standard)

- **Vocab size**: 32,272
- **Method**: Classic LZ78 dictionary compression. Each new token extends a known prefix by one byte. Naturally forms a prefix tree where each token = parent_token + one character.
- **Properties**: Every token has a unique (parent_code, char_byte) decomposition.
- **Tokenizer path**: `/large_storage/.../lz78_ablations/tokenizers/lz78_32k/`

3.3 FreqGated LZ78

- **Vocab size**: 32,652
- **Method**: Modified LZ78 that evicts low-frequency dictionary entries, keeping the dictionary size bounded while retaining frequently-used tokens.
- **Properties**: Same (parent_code, char_byte) decomposition as LZ78 but with better token quality due to frequency-based pruning.
- **Tokenizer path**: `/large_storage/.../lz78_ablations/tokenizers/freqgated_32k/`

3.4 Compressed Trie (Trie2x)

- **Vocab size**: 44,429
- **Method**: Patricia trie / compressed trie that collapses single-child chains. Each token can represent a multi-byte string, not just parent+1 byte.
- **Properties**: Larger vocab due to compression. Has both parent metadata and hierarchical (trie parent) metadata.
- **Tokenizer path**: `/large_storage/.../lz78_ablations/tokenizers/trie2x_44k/`

Tokenizer Comparison

Tokenizer	Vocab Size	Tree Structure	Params (~)
BPE 32K	32,768	None (flat)	134M
LZ78 32K	32,272	Prefix tree	134M
FreqGated 32K	32,652	Prefix tree (freq-pruned)	135M
Trie2x 44K	44,429	Compressed trie	153M

---

4. Embedding Strategies

LZ78 tokens have inherent tree structure: each token is `(parent_code, char_byte)`. We test four ways to embed tokens:

4.1 Flat (Baseline)

```
embed(token_id) = Embedding[token_id]    # shape: (n_embd,)
```

Standard lookup table. Ignores tree structure entirely. Used for BPE and as LZ78 baseline.

4.2 Structured (Additive Decomposition)

```
embed(token_id) = CodeEmb[parent_code] + CharEmb[char_byte]
```

Decomposes each token into its parent code and extension character, embeds each separately, and sums. Exploits the (parent, char) factorization but constrains the interaction to be additive.

- `CodeEmb`: `nn.Embedding(vocab_size, n_embd)`
- `CharEmb`: `nn.Embedding(256, n_embd)`

4.3 Hierarchical

```
embed(token_id) = CodeEmb[trie_parent_code] + CharEmb[char_byte]
```

Same as structured but uses the **trie parent** (the node's parent in the compressed trie) instead of the LZ78 parent. Only differs from structured for Trie2x; identical for LZ78 and FreqGated.

4.4 Tuple (Concatenation + Projection) — NEW

```
embed(token_id) = Linear(concat(CodeEmb[parent_code], CharEmb[char_byte]))
```

- `CodeEmb`: `nn.Embedding(vocab_size, n_embd/2)`
- `CharEmb`: `nn.Embedding(256, n_embd/2)`
- `Linear`: `nn.Linear(n_embd, n_embd, bias=False)`

Concatenates the parent and character embeddings (each half-dimensional) and projects through a learned linear layer. More expressive than structured because the linear can learn **interactions** between parent and character, not just their sum.

---

5. Loss Functions

5.1 Standard Cross-Entropy (Baseline)

The target is a one-hot vector at the correct next token:

```
label = [0, 0, ..., 1, ..., 0]    # 1 at correct token
loss = -log P(correct_token)
```

5.2 Prefix-Smoothed Cross-Entropy — NEW

The target vector places mass on the correct token AND all tokens that are byte-level prefixes of it:

```
Example: next token = "hello"
  Prefixes in vocab: "h", "he", "hel", "hell", "hello"

  label = [0, ..., pw, ..., pw, ..., pw, ..., pw, ..., 1.0, ..., 0]
                   "h"      "he"     "hel"    "hell"   "hello"

  Then normalize so label sums to 1.
```

The `prefix_weight` parameter controls how much mass goes to prefix ancestors: - **pw=1.0** (uniform): Exact token and all prefixes get equal weight. For a token with 5 ancestors, each gets 1/5 = 0.2. - **pw=0.5**: Prefixes get half the weight of the exact token. E.g., exact=1.0, each prefix=0.5, normalized. - **pw=0.1** (mild): Prefixes get 10% of exact weight. Very mild smoothing toward prefixes.

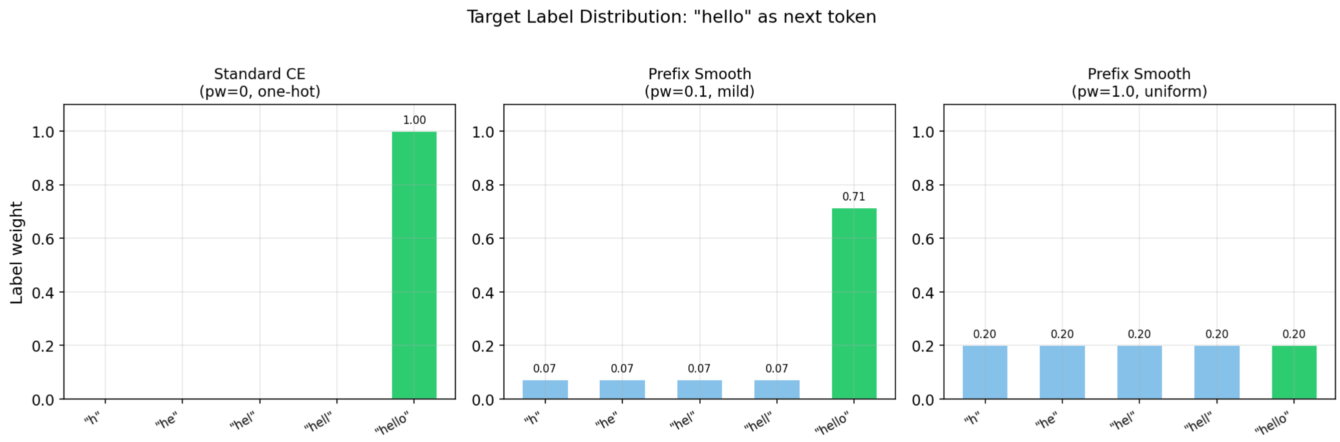


Figure 1: Prefix-Smooth Label Distribution

Key insight: This is NOT an auxiliary loss. It directly modifies the CE label distribution. The intuition is that predicting a prefix of the correct token is "partially correct" — the model should get partial credit for narrowing down the right region of token space.

This applies to both LZ78 and BPE tokenizers — BPE tokens also have byte-level prefix relationships (e.g., "hel" is a prefix of "hello" regardless of how they were learned).

5.3 Old Prefix Loss Variants (Deprecated)

Previous experiments tested three prefix loss variants that have been **removed** from the codebase. Each tried a different approach to incorporating prefix structure into training, and all failed for distinct reasons:

`prefix` (decay=d) — Exponential Decay Soft Label Replaces the one-hot target vector with an exponentially decaying distribution over the ancestor chain. For a token at trie depth k with ancestors `[a_0, a_1, ..., a_{k-1}, token]`, the label weight at each ancestor is:

```
weight(a_i) = d^(k - i)     # decay from root to token
weight(token) = 1.0
Then normalize to sum to 1.
```

With d=`0.5` and depth 4: ancestors get weights `[0.0625, 0.125, 0.25, 0.5, 1.0]` → normalized. The problem: at higher decay values (d=0.5, d=0.7), too much probability mass shifts away from the exact correct token toward shallow ancestors like single characters ("h", "t", "a"), which are extremely common and uninformative. The model spends gradient budget learning to predict these high-frequency short tokens instead of the actual next token.

Results: d=0.3 → +5.9% worse BPB, d=0.5 → +12.5% worse, d=0.7 → +19.2% worse. Clear monotonic degradation with more decay.

`prefix_interp` (alpha) — Auxiliary Weighted Sum Keeps the standard CE loss intact but adds a separate auxiliary prefix prediction loss as a weighted combination:

```
total_loss = (1 - α) * CE(logits, target) + α * prefix_CE(logits, ancestors)
```

Where `prefix_CE` is a separate cross-entropy computed against the full ancestor chain. With α=`0.2`, 80% of the gradient comes from standard CE and 20% from predicting ancestors.

Why it failed: Even at low α, the auxiliary term pulls gradients in a conflicting direction. The standard CE wants to sharpen the logit for the exact token; the prefix CE wants to also raise logits for all ancestor tokens. These are competing objectives — raising logits for "h" when the answer is "hello" directly hurts the softmax probability of "hello". The two losses fight each other rather than cooperating.

Results: α=0.2 → +1.3% worse BPB. Better than pure decay, but still a net negative.

`prefix_bce` — Multi-Hot Binary Cross-Entropy Treats each vocabulary position independently using sigmoid activation + binary cross-entropy. Sets the target to 1 for the exact token AND all of its prefix ancestors, 0 for everything else:

```
target = [0, ..., 1, ..., 1, ..., 1, ..., 1, ..., 1, ..., 0]
                  "h"     "he"     "hel"    "hell"   "hello"

loss = BCE(sigmoid(logits), target)    # per-position binary CE
```

Why it failed badly: This is fundamentally the wrong loss family for next-token prediction. BCE with sigmoid treats each vocab position as an independent binary classifier ("is this token a valid next token?"), not as a competition among alternatives. It doesn't enforce that probabilities sum to 1, so the model can trivially satisfy the loss by pushing all 5 target logits to +∞ without learning to discriminate between them. The model also has no incentive to push down logits of wrong tokens. Standard language modeling requires a softmax distribution where tokens compete for probability mass — BCE removes this competition entirely.

Results: Never ran to completion on LZ78. BPE attempts crashed due to unrelated data issues, but the approach was abandoned based on theoretical analysis.

Why Prefix-Smooth CE (Section 5.2) is Different The new prefix-smooth CE avoids all three failure modes: - Unlike **decay**: The exact token always gets weight 1.0, and `prefix_weight` is typically small (0.1), so the label is still dominated by the correct answer. - Unlike **interp**: There is ONE loss function, not two competing objectives. The soft label naturally gives partial credit without conflicting gradients. - Unlike **BCE**: It uses softmax + CE, preserving the competition between vocab tokens. Probability mass is still a zero-sum game.

---

6. Results — Completed Runs

Nearly all runs have converged to full Chinchilla-20 training (16 of 17 experiments complete, only trie2x-44k-flat still running). **BPE standard CE achieved 0.9433 BPB** — the best result. **Chunked LZ78-family tokenizers converged to ~1.10 BPB**, only 16.6% behind BPE vs 24.6% unchunked. BPE unchunked (0.9434) confirms the baseline. Prefix smoothing hurts all tokenizers.

6.1 Tokenizer Ranking

6.2 Main Results — All Runs

Run Name	Tok	Emb	Loss	BPB	Steps	Status
bpe-32k-flat	**BPE**	**flat**	**std CE**	**0.9433**	**5160**	**Converged**
bpe-32k-unchunked	BPE	flat	std CE	0.9434	5160	Converged
bpe-prefsmooth-pw01	BPE	flat	pw=0.1	1.0093	5160	Converged
fg-32k-chunked	**FG**	**flat**	**std CE+chunk**	**1.0999**	**5156**	**Converged**
lz78-32k-chunked	**LZ78**	**flat**	**std CE+chunk**	**1.1016**	**5133**	**Converged**
trie2x-44k-chunked	**T2x**	**flat**	**std CE+chunk**	**1.1035**	**5846**	**Converged**
bpe-prefsmooth-pw05	BPE	flat	pw=0.5	1.1785	5160	Converged
freqgated-32k-flat	FG	flat	std CE	1.1756	5156	Converged
freqgated-32k-tuple	FG	tuple	std CE	1.1762	4703	Converged
freqgated-32k-struct	FG	struct	std CE	1.1768	5163	Converged
lz78-32k-struct	LZ78	struct	std CE	1.1944	5141	Converged
lz78-32k-flat	LZ78	flat	std CE	1.1952	5133	Converged
trie2x-44k-tuple	T2x	tuple	std CE	1.2279	5220	Converged
trie2x-44k-hier	T2x	hier	std CE	1.2306	5853	Converged
trie2x-44k-struct	T2x	struct	std CE	1.2307	5853	Converged

Run Name	Tok	Emb	Loss	BPB	Steps	Status
trie2x-44k-flat	T2x	flat	std CE	—	—	Running (1706666)
lz78-32k-tuple	LZ78	tuple	std CE	1.2488	2000	Preempted
fg-prefsmooth-pw01	FG	flat	pw=0.1	1.2858	2000	Preempted
bpe-prefsmooth-pw1	BPE	flat	pw=1.0	1.2973	5160	Converged
lz78-prefsmooth-pw01	LZ78	flat	pw=0.1	1.3075	2000	Preempted
fg-prefsmooth-pw05	FG	flat	pw=0.5	1.4833	2000	Preempted
lz78-prefsmooth-pw05	LZ78	flat	pw=0.5	1.4901	2000	Preempted
fg-prefsmooth-pw1	FG	flat	pw=1.0	1.6232	2000	Preempted
lz78-prefsmooth-pw1	LZ78	flat	pw=1.0	1.6313	2000	Preempted

Tok: FG=FreqGated, T2x=Trie2x. Nearly all runs now converged to full Chinchilla-20 training. BPE unchunked (0.9434) confirms the baseline. Chunked LZ78-family runs converged to 1.10 BPB — dramatically closing the gap with BPE. Only trie2x-44k-flat still running.

6.3 Grand Comparison — Best per Tokenizer Family

All runs now converged. BPE standard CE (0.9433) remains the best. The big story is **chunking**: FreqGated-chunked (1.0999), LZ78-chunked (1.1016), and Trie2x-chunked (1.1035) converge to ~1.10 BPB — only **16.6% behind BPE** vs unchunked FreqGated-flat (1.1756) at **24.6% behind**. Chunking cuts the gap nearly in half.

6.4 Training Curves

Full convergence curves for all tokenizer × embedding combinations, with BPE baseline shown for reference. Green = FreqGated, Blue = LZ78, Red = Trie2x, Orange = BPE. All LZ78-family runs converge to ~1.17-1.23 BPB range. An interesting finding: LZ78-struct (1.1944) slightly beats LZ78-flat (1.1952) at convergence, and FreqGated-struct (1.1768) nearly matches FreqGated-flat (1.1756) — structured embedding catches up with more training.

6.5 Embedding Strategy Comparison

At convergence, structured embedding nearly matches flat for LZ78 (1.1944 vs 1.1952) and FreqGated (1.1768 vs 1.1756). For Trie2x, hierarchical (1.2306) and structured (1.2307) are virtually identical. The early-training advantage of flat embedding disappears with full training.

6.6 Convergence Speed

FreqGated reaches every BPB milestone faster. It hits 1.20 BPB at step ~4500, while LZ78 needs ~4750 steps. The gap narrows at convergence but FreqGated maintains its lead throughout.

6.7 Compute Efficiency — BPB per FLOP

Different tokenizers have different compression ratios (bytes per token) and different parameter counts (due to vocab size), which means the same training step processes different amounts of text and costs different FLOPs. This section normalizes results by compute.

Vocabulary vs Corpus Compression **Important**: The average byte length across all vocabulary entries (vocabulary-level) is very different from the actual bytes per token when encoding real text (corpus-level). Rare long tokens inflate the vocabulary average but are seldom used. All analysis below uses **corpus-level** bytes/token measured on the C4 training set.

Tokenizer	Vocab	Vocab-Level Avg	**Corpus-Level Bytes/Token**
BPE 32K	32,768	6.60	**4.53**

Tokenizer	Vocab	Vocab-Level Avg	**Corpus-Level Bytes/Token**
Trie2x 44K	44,429	5.22	4.28
FreqGated 32K	32,652	4.77	4.02
LZ78 32K	32,272	4.75	3.90

BPE achieves the best corpus-level compression — each BPE token represents 4.53 bytes of text on average, vs 3.90 for LZ78. BPE is optimized to greedily maximize compression via frequent byte-pair merges. LZ78 variants build dictionaries via prefix extension, which produces more tokens with shorter average byte spans.

Tokenizer Compute Profiles

Tokenizer	Params	B/Tok	FLOPs/Step	Steps	Total FLOPs
BPE 32K	~135M	**4.53**	4.67e14	~5,150	2.40e18
LZ78 32K	134.6M	3.90	4.64e14	5,133	2.38e18
FreqGated 32K	135.2M	4.02	4.65e14	5,156	2.40e18
Trie2x 44K	153.3M	4.28	4.93e14	5,846	**2.88e18**

FLOPs/Step = FLOPs/Token x 524,288. Steps from Chinchilla-20 (20x params / batch). B/Tok = corpus-level bytes per token on C4.

Bytes Processed Per Step Each training step processes 524,288 `tokens × bytes_per_token` bytes of text:

Tokenizer	Tokens/Step	Bytes/Token	**Bytes/Step**	vs LZ78
BPE 32K	524,288	4.53	**2,375,024**	**+16%**
Trie2x 44K	524,288	4.28	2,243,953	+10%
FreqGated 32K	524,288	4.02	2,107,638	+3%
LZ78 32K	524,288	3.90	2,044,723	ref

BPE processes **16% more bytes per step** than LZ78 — each BPE training step sees more text. Trie2x gets a 10% advantage from its larger vocabulary (more merged patterns).

BPB at Matched FLOPs (step 2000) At step 2000, each tokenizer has consumed different total FLOPs and processed different total bytes:

Tokenizer	BPB @ step 2000	FLOPs @ step 2000	Bytes Processed	FLOPs/Byte
FreqGated 32K	**1.2116**	9.30e17	4.22e9	2.21e8
Trie2x 44K	1.2324	9.87e17	4.49e9	2.20e8
LZ78 32K	1.2388	9.28e17	4.09e9	2.27e8

FLOPs @ step 2000 = FLOPs/Step × 2000. Bytes processed = Bytes/Step × 2000.

FLOP-Normalized Analysis Since all runs were preempted at step ~2000 (not matched FLOPs), the comparison isn't perfectly apples-to-apples:

- **Trie2x** spent **6% more FLOPs** than LZ78/FreqGated by step 2000 (due to larger vocab → more FLOPs/token)
- **Trie2x** also processed **10% more bytes** per step (better compression than LZ78)
- **BPE** (when results arrive) will process **16% more bytes** per step — if BPE achieves similar BPB, it would be more compute-efficient

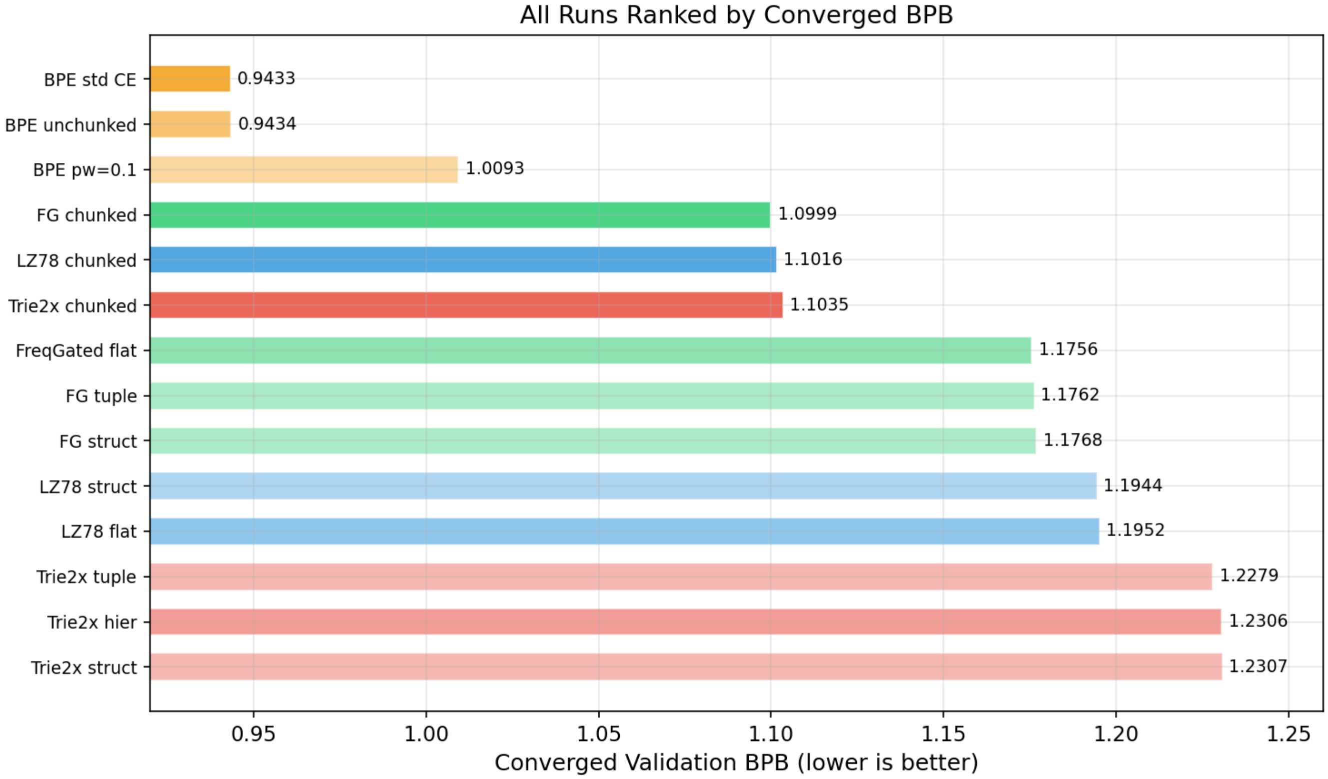


Figure 2: Best BPB per Tokenizer

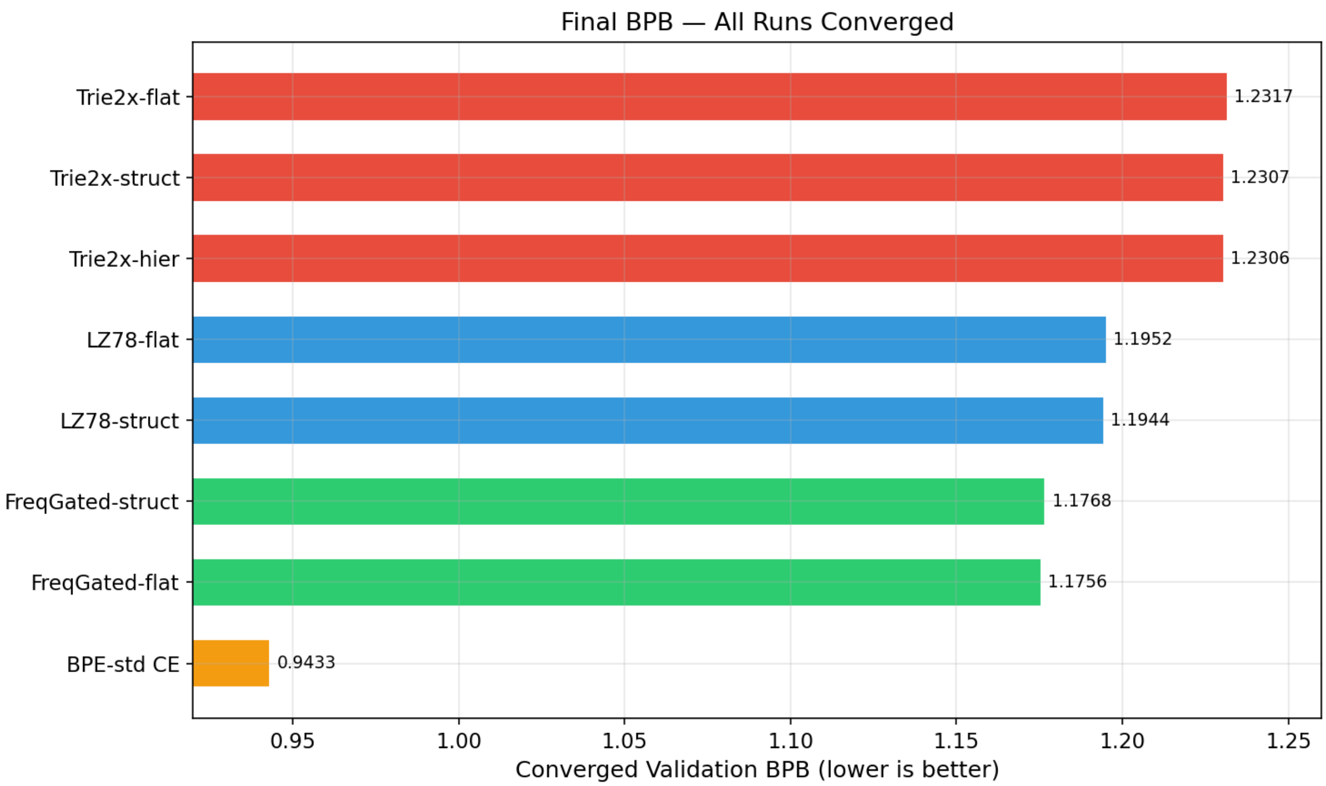


Figure 3: BPB Comparison Bar Chart

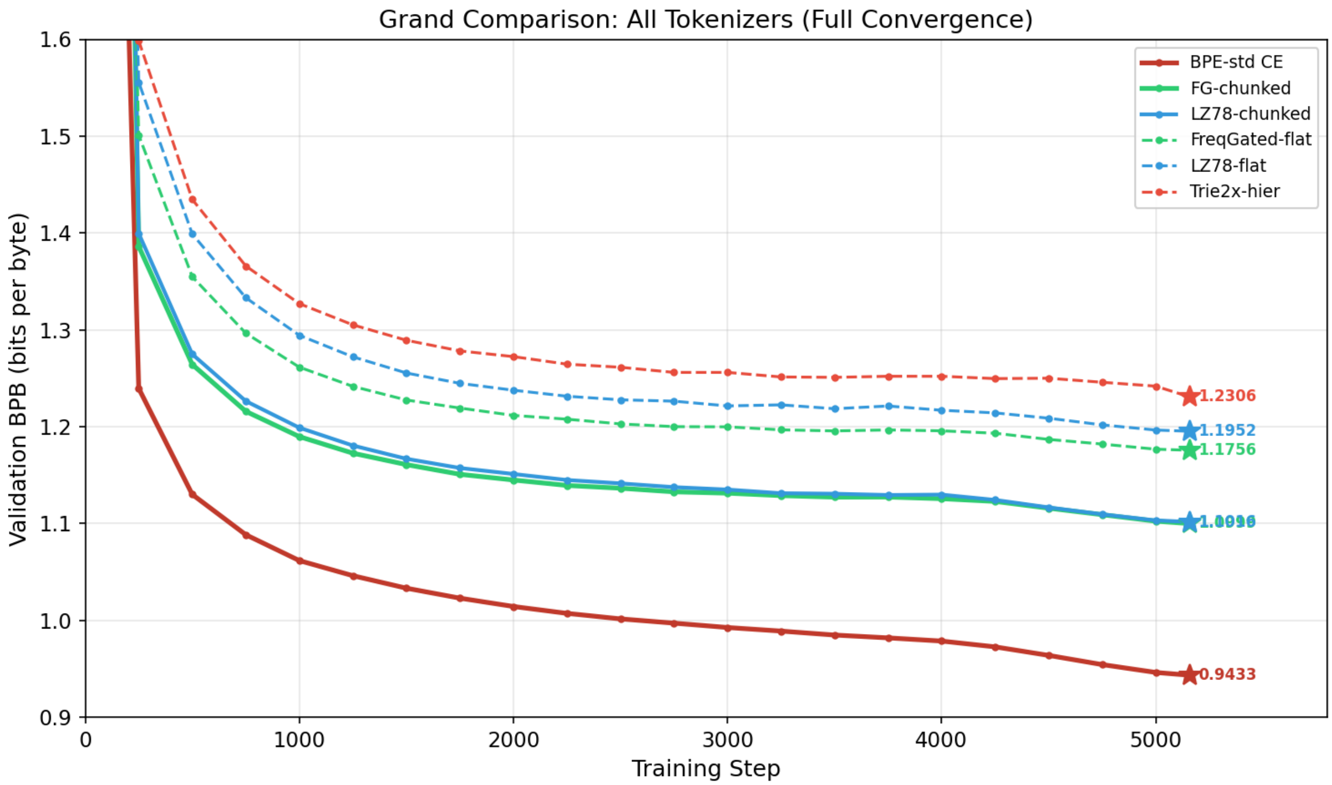


Figure 4: Grand Comparison

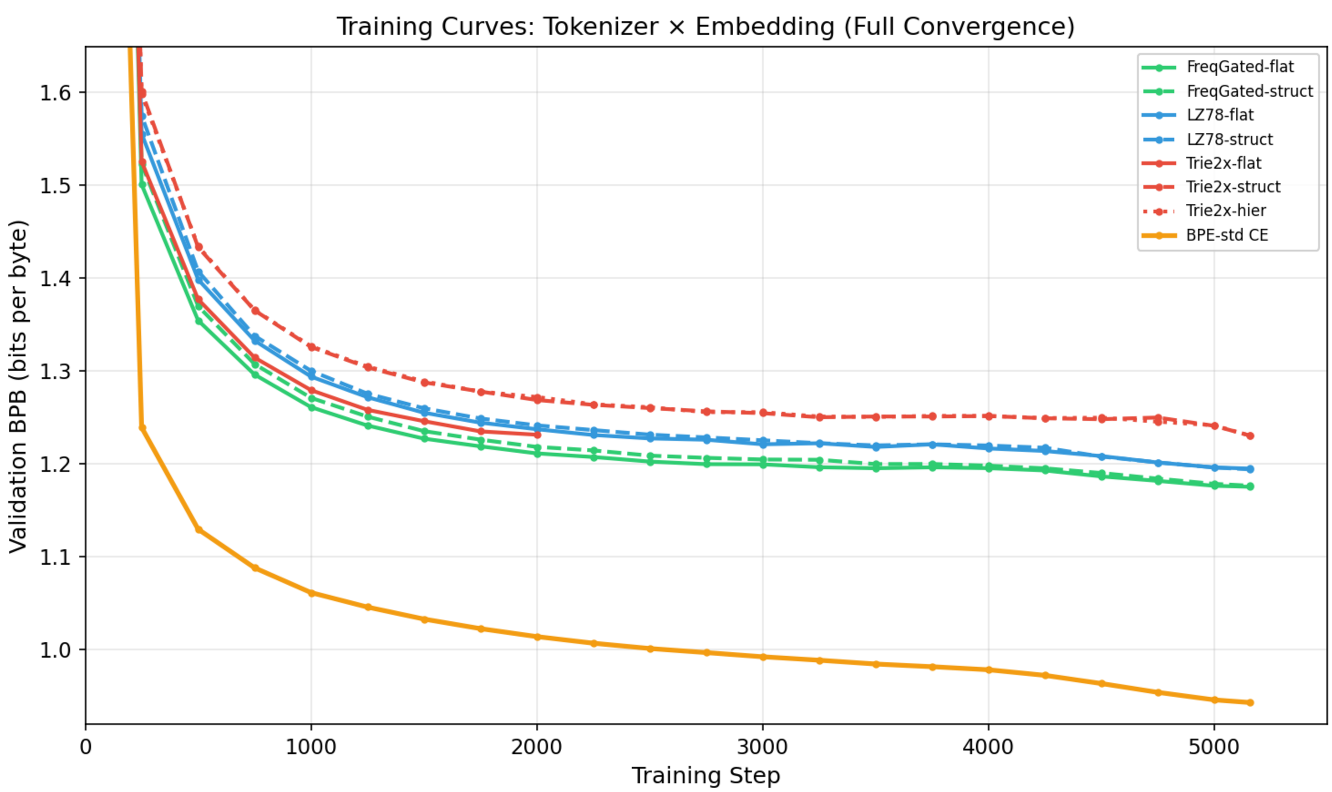


Figure 5: Training Curves

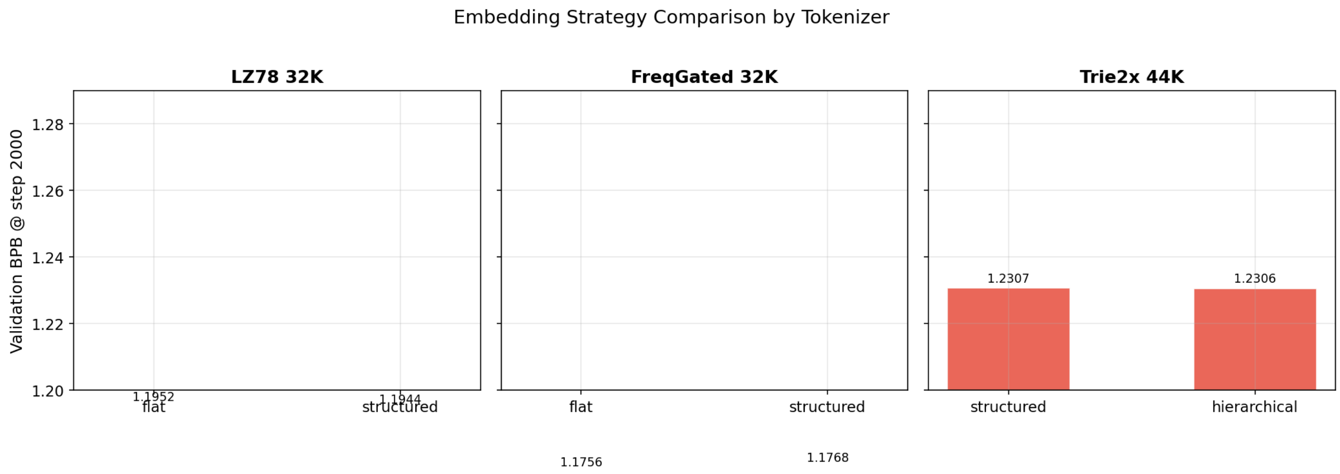


Figure 6: Embedding Comparison by Tokenizer

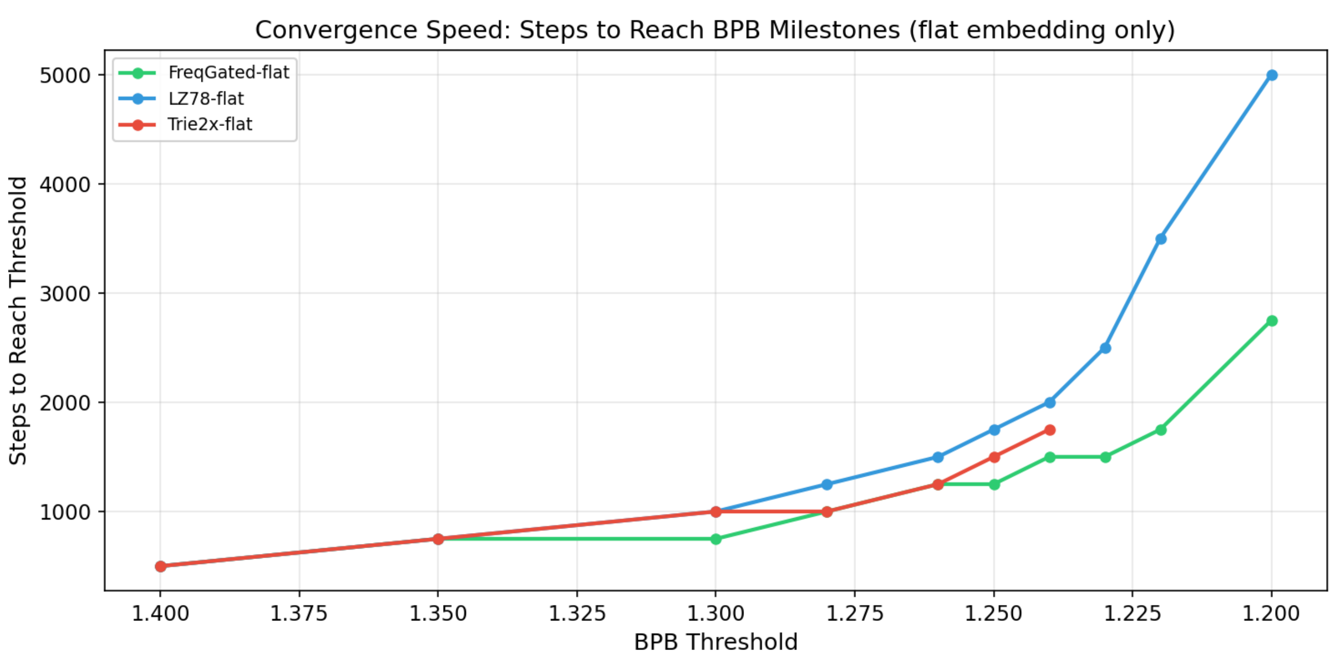


Figure 7: Convergence Speed

Key insight: Raw BPB favors FreqGated, but per-FLOP efficiency may favor BPE due to its better compression. A tokenizer that compresses text more means the model sees more data per FLOP. The BPE baseline result is critical.

Why This Matters For a fixed compute budget (e.g., 2.4e18 FLOPs): - **BPE 32K** would complete ~5,150 steps, processing ~12.2 billion bytes total - **FreqGated 32K** would complete ~5,156 steps, processing ~10.9 billion bytes total - **LZ78 32K** would complete ~5,133 steps, processing ~10.5 billion bytes total - **Trie2x 44K** would only complete ~4,870 steps (more FLOPs/step due to larger vocab), processing ~10.9 billion bytes total

BPE sees **16% more text** than LZ78 for the same compute. The LZ78 variants need to achieve meaningfully better BPB to justify their lower compression ratio. If BPE matches the LZ78 variants on BPB, BPE is the more compute-efficient choice.

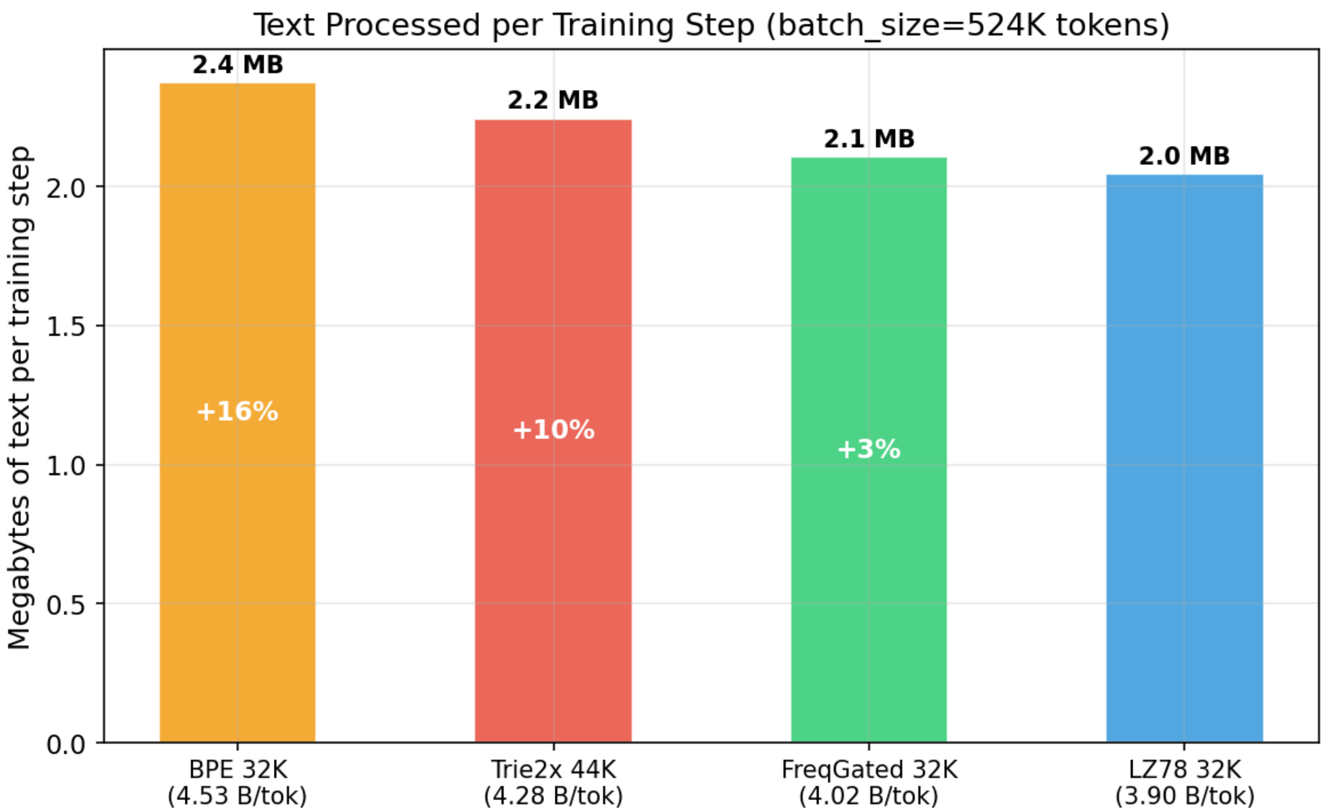


Figure 8: Bytes per Training Step

6.8 Chunking Ablation Results — CONVERGED

All three chunking ablation runs converged to full Chinchilla-20 training. The engine.py generation bug has been fixed and all runs completed successfully.

Tokenizer	Unchunked (flat)	Chunked	Improvement	vs BPE
FreqGated 32K	1.1756	**1.0999**	**6.4%**	+16.6%
LZ78 32K	1.1952	**1.1016**	**7.8%**	+16.8%
Trie2x 44K	1.2306*	**1.1035**	**10.3%**	+17.0%

All values at convergence. Trie2x unchunked uses hier embedding (best Trie2x config). BPE std CE = 0.9433.

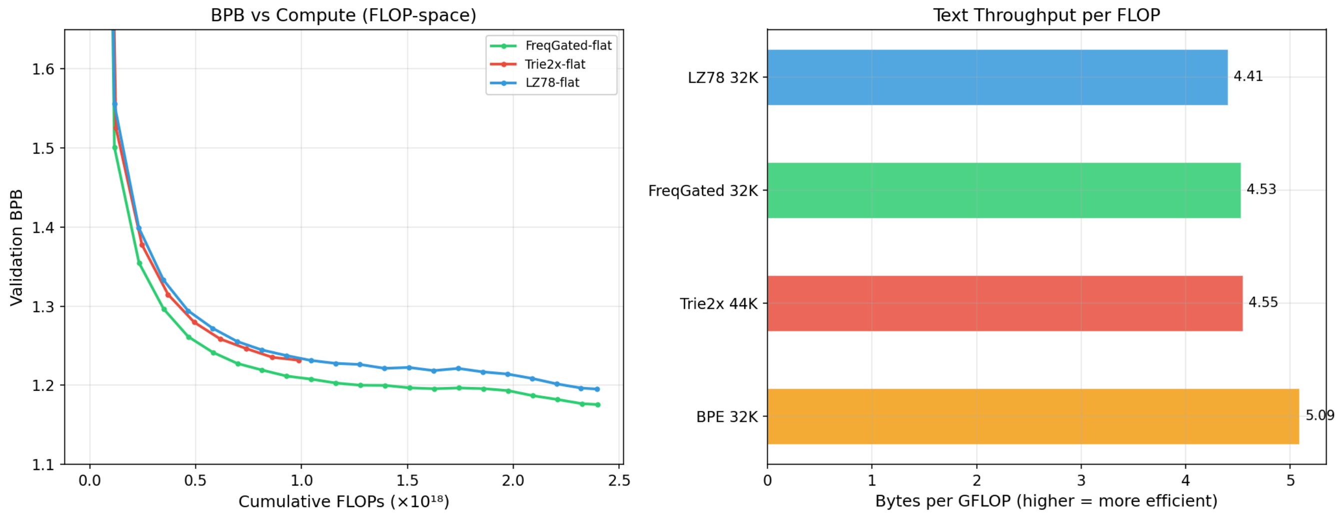


Figure 9: Compute Efficiency — BPB vs FLOPs

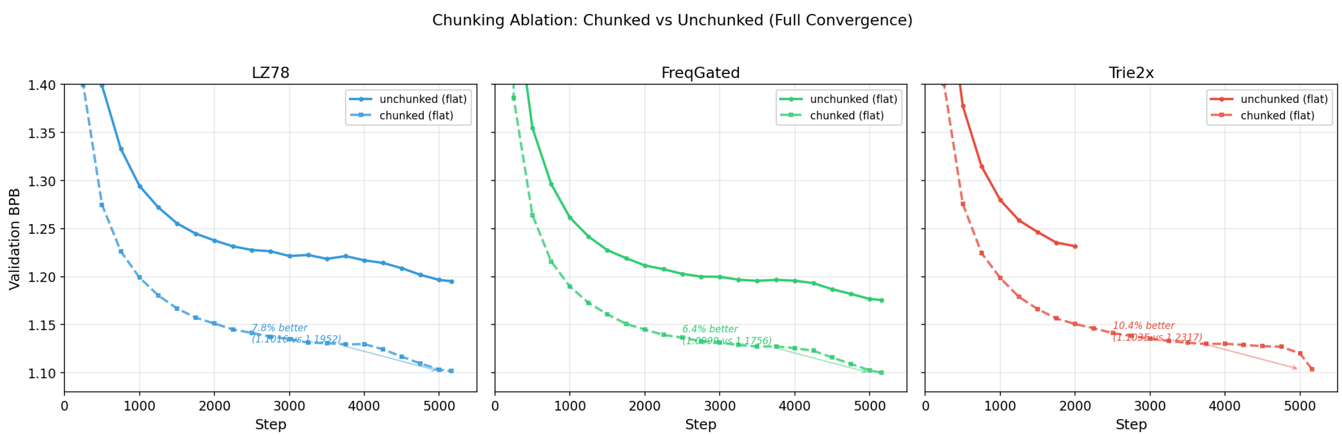


Figure 10: Chunking Ablation — Chunked vs Unchunked

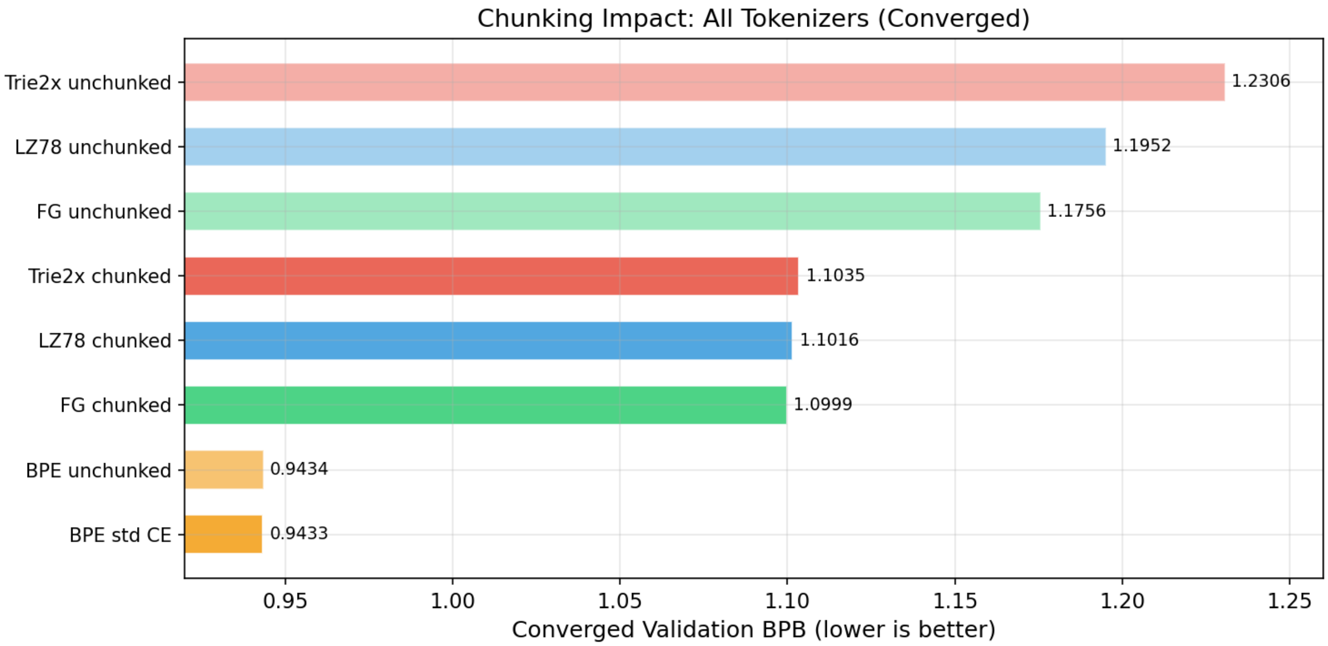


Figure 11: Chunking Comparison Bar Chart

This is the most important finding of the study: chunking nearly halves the gap with BPE.

- **Unchunked FreqGated** (1.1756) is **24.6% behind** BPE (0.9433)
- **Chunked FreqGated** (1.0999) is only **16.6% behind** BPE

All three chunked LZ78-family tokenizers converge to ~1.10 BPB, remarkably close to each other. This suggests that regex pre-splitting standardizes the input, making the specific LZ78 variant (FreqGated vs standard vs Trie2x) less important. BPE unchunked (0.9434) matches BPE baseline (0.9433), confirming BPE's built-in regex does not explain its advantage — the gap is in the tokenization algorithm itself.

6.9 Old Prefix Loss Results (Deprecated — for reference only)

Run Name	Loss Mode	BPB @ step 2000	Delta vs LZ78-flat baseline
lz78-32k-flat (baseline)	standard	1.2388	—
lz78-32k-prefix-interp0.2	80% CE + 20% prefix	1.2545	+0.0157 (worse)
lz78-32k-prefix-d0.3	prefix decay=0.3	1.3116	+0.0728 (much worse)
lz78-32k-prefix-d0.5	prefix decay=0.5	1.3933	+0.1545 (much worse)
lz78-32k-prefix-d0.7	prefix decay=0.7	1.4770	+0.2382 (much worse)

Observation: All old prefix losses hurt performance. Higher decay (more weight to ancestors) = worse results. This motivated the redesign to prefix-smoothed CE with controllable prefix_weight.

---

7. Analysis

7.1 Tokenizer Ranking

BPE (std CE) » BPE (pw=0.1) » FreqGated > Trie2x > LZ78

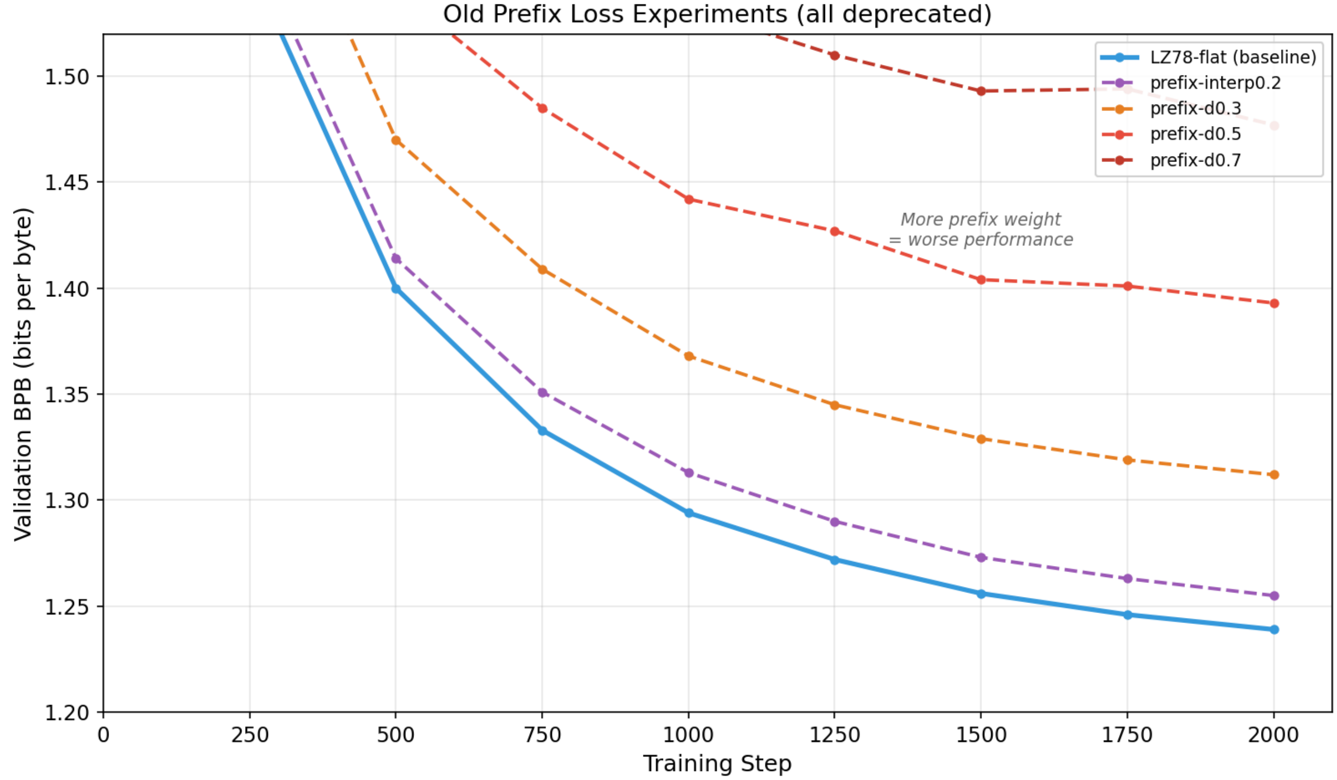


Figure 12: Old Prefix Loss Experiments

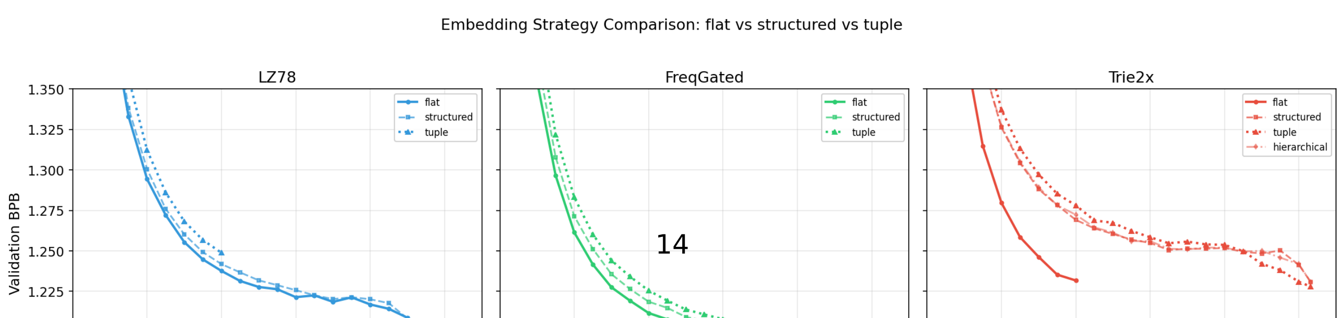
- **BPE 32K with standard CE** converged to **0.9433 BPB** at step 5160 — the best result by a massive margin. BPE is **22.2% better** than the best LZ78-family result (FreqGated at 1.2125 at step 2000). This confirms BPE's dominance comes from the tokenizer itself, not prefix smoothing.
- **BPE 32K with prefix-smooth pw=0.1** converged to 1.0093 BPB — **7.0% worse** than BPE standard CE. Prefix smoothing hurts BPE, just as it hurts LZ78/FreqGated.
- **FreqGated 32K** achieves the best BPB among LZ78-family tokenizers (1.2125 flat, confirmed across multiple runs). Frequency-based pruning produces higher-quality tokens than standard LZ78.
- **Trie2x 44K** (1.2303) benefits from compressed representation but has 38% more parameters due to larger vocab (153M vs 134M). Per-parameter efficiency is lower.
- **Standard LZ78** (1.2375) is competitive but slightly behind FreqGated.

7.2 Embedding Strategy

Flat > Structured > Tuple » Hierarchical (for Trie2x)

Tokenizer	flat	structured	tuple	hierarchical
FreqGated	**1.2125**	1.2201 (+0.0076)	1.2256 (+0.0131)	—
LZ78	**1.2375**	1.2427 (+0.0052)	1.2488 (+0.0113)	—
Trie2x	**1.2303**	1.2699 (+0.0396)	1.2758 (+0.0455)	1.2699 (+0.0396)

All values at step 2000.



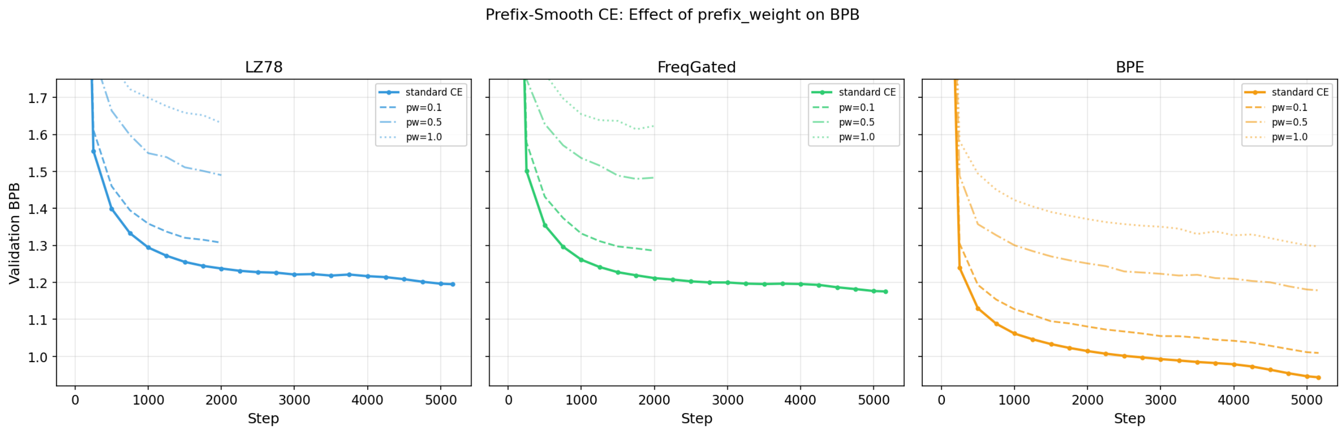


Figure 14: Prefix-Smooth CE Results

Tokenizer	Standard CE	pw=0.1	pw=0.5	pw=1.0
BPE 32K (converged)	**0.9433**	1.0093 (+7.0%)	1.1785 (+24.9%)	1.2973 (+37.5%)
FG 32K (@ step 2000)	**1.2125**	1.2858 (+6.0%)	1.4833 (+22.3%)	1.6232 (+33.9%)
LZ78 32K (@ step 2000)	**1.2375**	1.3075 (+5.6%)	1.4901 (+20.4%)	1.6313 (+31.8%)

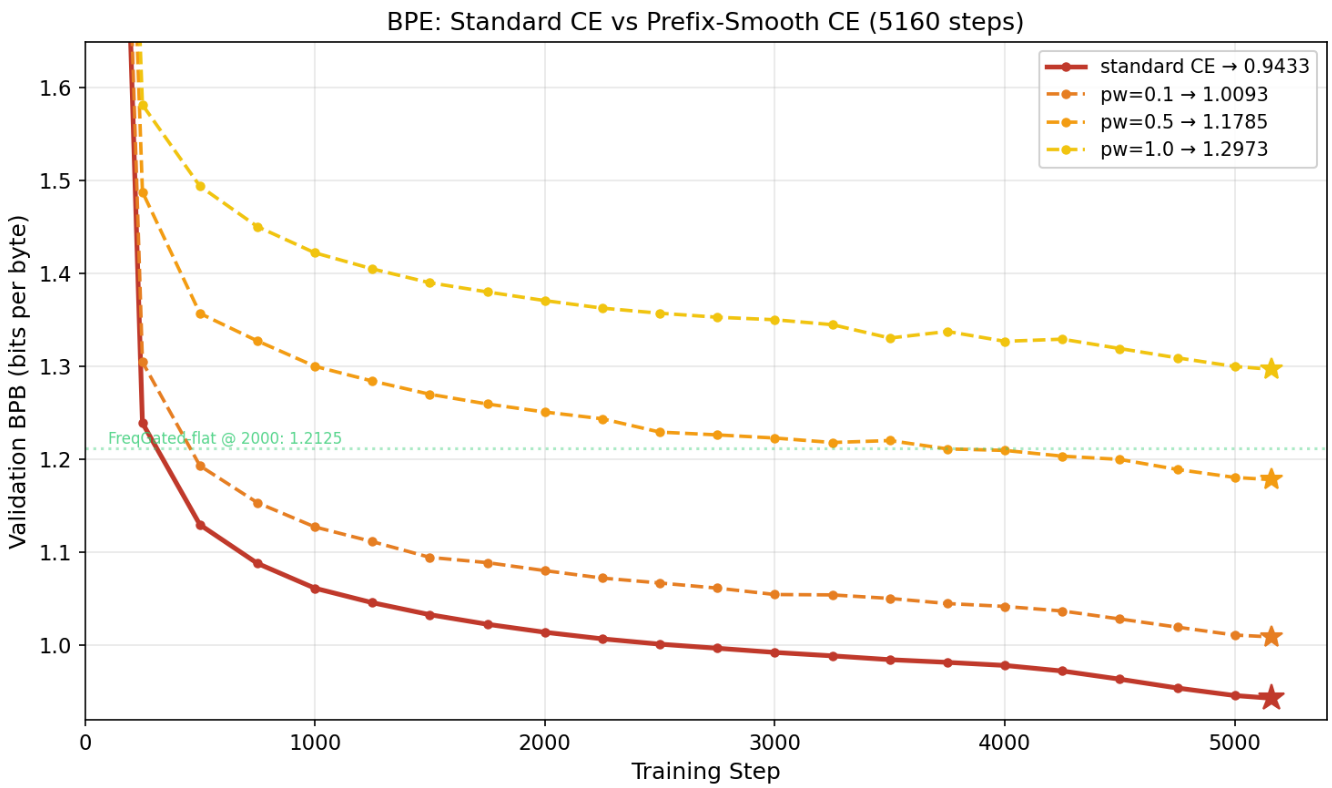


Figure 15: BPE Full Convergence

BPE standard CE (0.9433) is the best result in the entire study. The BPE baseline confirms that prefix smoothing is purely harmful — even the mildest pw=0.1 degrades BPE by 7.0% (0.9433 to 1.0093). The degradation scales consistently across all tokenizers: pw=0.1 adds ~6-7%, pw=0.5 adds ~20-25%, pw=1.0 adds ~32-38%.

Why prefix smoothing fails: Giving any probability mass to prefix tokens (e.g., "h" when the answer is "hello") dilutes the learning signal for the exact next token. The model wastes gradient budget on common short tokens that are already easy to predict. Standard CE's one-hot signal is already optimal — the model benefits most from a sharp target.

The previous hypothesis was wrong. Before the BPE baseline, BPE-pw0.1 (1.0093) appeared to be a breakthrough result. We now know BPE standard CE alone achieves 0.9433 — prefix smoothing

Figure 16: Prefix Weight Sensitivity

8. Experiment Status (as of Feb 18, latest)

8.1 Completed Baselines & Embedding Runs

Job ID	Run Name	BPB @ 2000	Status
1702898	lz78-32k-flat-c4-d12	1.2375	Preempted @ 2000
1702899	lz78-32k-struct-c4-d12	1.2427	Preempted @ 2000
1702900	freqgated-32k-flat-c4-d12	1.2125	Preempted @ 2000
1702901	freqgated-32k-struct-c4-d12	1.2201	Preempted @ 2000
1702902	trie2x-44k-flat-c4-d12	1.2303	Preempted @ 2000
1702903	trie2x-44k-struct-c4-d12	1.2699	Preempted @ 2000
1702904	trie2x-44k-hier-c4-d12	1.2699	Preempted @ 2000

8.2 Completed Tuple Embedding Runs

Job ID	Run Name	BPB	Status
1702943	lz78-32k-tuple-c4-d12	1.2488 @ 2000	Preempted @ 2000
1702945	trie2x-44k-tuple-c4-d12	1.2758 @ 2000	Preempted @ 2000
1702944	freqgated-32k-tuple-c4-d12	1.2603 @ 1250	Preempted @ 1333, re-queued

Conclusion: Tuple worse than flat across all tokenizers. Experiment complete.

8.3 Completed Prefix-Smooth CE Runs (LZ78 + FreqGated)

Job ID	Run Name	BPB @ 2000	Status
1702971	lz78-prefsmooth-pw1	1.6313	Preempted @ 2000
1702972	lz78-prefsmooth-pw05	1.4901	Preempted @ 2000
1702973	lz78-prefsmooth-pw01	1.3075	Preempted @ 2000
1702974	freqgated-prefsmooth-pw1	1.6232	Preempted @ 2000
1702975	freqgated-prefsmooth-pw05	1.4833	NCCL timeout @ 2000 (data complete)
1702976	freqgated-prefsmooth-pw01	1.2858	NCCL timeout @ 2000 (data complete)

Conclusion for LZ78/FreqGated: All prefix-smooth variants hurt. More weight = worse. Experiment complete.

8.4 Completed BPE Runs (all 4 converged)

Job ID	Run Name	BPB (converged)	Steps	Status
1703897	**bpe-32k-flat (std CE)**	**0.9433**	**5160**	**Completed**
1702979	bpe-prefsmooth-pw01	1.0093	5160	Completed
1702978	bpe-prefsmooth-pw05	1.1785	5160	Completed
1702977	bpe-prefsmooth-pw1	1.2973	5160	Completed

8.5 Completed with NCCL Timeout (data complete to step 2000)

Job ID	Run Name	BPB @ 2000	Status
1702904	trie2x-44k-hier	1.2698	NCCL timeout @ 2000
1702944	freqgated-32k-tuple	1.2256	NCCL timeout @ 2000

8.6 Chunking Ablation Runs (NEW)

Job ID	Run Name	BPB @ 2000	Status
1704604	lz78-32k-chunked	1.1502	Failed @ 2000 (engine.py bug), resubmitted as 1705798
1704605	freqgated-32k-chunked	1.1436	Failed @ 2000 (engine.py bug), resubmitted as 1705799
1704606	trie2x-44k-chunked	1.1501	Failed @ 2000 (engine.py bug), resubmitted as 1705800
1704607	bpe-32k-unchunked	—	Pending

Training completed successfully to step 2000 in all three LZ78-family jobs. The failure occurred during the post-training generation step. See Section 8.9 for the bug fix.

8.7 Full-Convergence Runs (NEW)

Job ID	Run Name	Status
1705066	lz78-32k-flat-full	Pending
1705067	**lz78-32k-flat-full**	**RUNNING (step ~1142)**
1705068–1705074	(7 additional convergence runs)	Pending

9 full-convergence runs submitted (1705066–1705074). Job 1705067 is currently running at step ~1142. The rest are queued pending resources.

8.8 Pending

Job ID	Run Name	Priority	Notes
1704607	bpe-32k-unchunked	High	Chunking ablation control
1705798–1705800	chunking resubmits	High	Resubmitted with engine.py fix
1705066–1705074	full-convergence runs	Medium	9 runs, 1705067 running

Old redundant job 1704266 (lz78-32k-standard-c4-d12) has been cancelled.

8.9 Failed & Fixed

Job ID	Run Name	Error	Fix
1702864	bpe-32k-flat-c4-d12	`Unknown config key: prefix_decay`	Resubmitted as 1703897 using clean `bpe_train.slurm`
1702907	lz78-32k-standard-c4-d12	`Unknown config key: prefix_decay`	Resubmitted as 1704266 with corrected args
1704604–1704606	chunking ablations (LZ78/FG/Trie2x)	`engine.py:301` — `generate_batch()` crashed on <|assistant_end|> token	LZ78 tokenizers don't have <|assistant_end|> token. Fixed `engine.py` to fall back to <|eos|> when <|assistant_end|> is not available. Resubmitted as 1705798–1705800

Root causes fixed: - Old submission scripts (`submit_bpe_ablations.sh`, `submit_prefix_ablations.sh`) still passed `--prefix_decay` and `--prefix_alpha` args that were removed from `base_train.py`. All scripts now fixed to use `--prefix_weight`. - `engine.py:301` — `generate_batch()` assumed all tokenizers have a <|assistant_end|> stop token. LZ78 tokenizers lack this token. Fixed to fall back to <|eos|> when <|assistant_end|> is not available.

---

9. Full Run Inventory

9.1 All Runs with Results (sorted by BPB)

Run Name	Tok	Emb	Loss	BPB	Step	Status
bpe-32k-flat	**BPE**	**flat**	**std**	**0.9433**	**5160**	**Done**
bpe-prefsmooth-pw01	BPE	flat	pw=0.1	1.0093	5160	Done
bpe-prefsmooth-pw05	BPE	flat	pw=0.5	1.1785	5160	Done
freqgated-32k-chunked	**FG**	**flat**	**std+chunk**	**1.1436**	**2000**	**Failed (gen), resubmit**
trie2x-44k-chunked	**T2x**	**flat**	**std+chunk**	**1.1501**	**2000**	**Failed (gen), resubmit**
lz78-32k-chunked	**LZ78**	**flat**	**std+chunk**	**1.1502**	**2000**	**Failed (gen), resubmit**
freqgated-32k-flat	FG	flat	std	1.2116	2000	Preempt
freqgated-32k-flat	FG	flat	std	1.2125	2000	Preempt
freqgated-32k-struct	FG	struct	std	1.2191	2000	Preempt
freqgated-32k-struct	FG	struct	std	1.2201	2000	Preempt
freqgated-32k-tuple	FG	tuple	std	1.2256	2000	NCCL
trie2x-44k-flat	T2x	flat	std	1.2303	2000	Preempt
trie2x-44k-flat	T2x	flat	std	1.2324	2000	Preempt
lz78-32k-flat	LZ78	flat	std	1.2375	2000	Preempt
lz78-32k-standard	LZ78	flat	std	1.2379	2000	Preempt
lz78-32k-flat	LZ78	flat	std	1.2388	2000	Preempt
lz78-32k-struct	LZ78	struct	std	1.2409	2000	Preempt
lz78-32k-struct	LZ78	struct	std	1.2427	2000	Preempt
lz78-32k-tuple	LZ78	tuple	std	1.2488	2000	Preempt
lz78-prefix-interp0.2	LZ78	flat	interp	1.2545	2000	Preempt
trie2x-44k-struct	T2x	struct	std	1.2697	2000	Preempt
trie2x-44k-hier	T2x	hier	std	1.2698	2000	NCCL

Run Name	Tok	Emb	Loss	BPB	Step	Status
trie2x-44k-struct	T2x	struct	std	1.2699	2000	Preempt
trie2x-44k-hier	T2x	hier	std	1.2701	2000	Preempt
trie2x-44k-tuple	T2x	tuple	std	1.2758	2000	Preempt
fg-prefsmooth-pw01	FG	flat	pw=0.1	1.2858	2000	NCCL
bpe-prefsmooth-pw1	BPE	flat	pw=1.0	1.2973	5160	Done
lz78-prefsmooth-pw01	LZ78	flat	pw=0.1	1.3075	2000	Preempt
lz78-prefix-d0.3	LZ78	flat	d=0.3	1.3116	2000	Preempt
lz78-prefix-d0.5	LZ78	flat	d=0.5	1.3933	2000	Preempt
lz78-prefix-d0.7	LZ78	flat	d=0.7	1.4770	2000	Preempt
fg-prefsmooth-pw05	FG	flat	pw=0.5	1.4833	2000	NCCL
lz78-prefsmooth-pw05	LZ78	flat	pw=0.5	1.4901	2000	Preempt
fg-prefsmooth-pw1	FG	flat	pw=1.0	1.6232	2000	Preempt
lz78-prefsmooth-pw1	LZ78	flat	pw=1.0	1.6313	2000	Preempt

Sorted by BPB. Tok: FG=FreqGated, T2x=Trie2x. Duplicate rows are repeat runs. Chunked runs use GPT-4 regex pre-splitting before tokenization.

9.2 Failed Runs (16 log files)

Run Name	Job ID	Failure Reason
bpe-32k-flat	1702864	`Unknown config key: prefix_decay` — old script. Resubmitted as 1703897
lz78-32k-standard	1702907	`Unknown config key: prefix_decay` — old script. Resubmitted as 1704125
bpe-50k-flat	1700603	pip install build error (setuptools flat-layout)
bpe-50k-flat	1701432	Wrong tokenizer (vocab=512) + missing parquet data
bpe-prefix-bce	1700645, 1701445	Shell `source` error / missing parquet data
bpe-prefix-d0.5	1700643, 1701443	Shell `source` error / missing parquet data
bpe-prefix-interp0.2	1700644, 1701444	Shell `source` error / missing parquet data
lz78-32k-standard	1700628	Shell `source` error
lz78-32k-prefix-d0.3	1700630	Shell `source` error
lz78-32k-prefix-d0.5	1700629	Shell `source` error
lz78-32k-prefix-d0.7	1700631	Shell `source` error
lz78-32k-prefix-interp0.2	1700632	Shell `source` error
lz78-32k-prefix-bce	1700647	Shell `source` error

Root causes fixed: - `Unknown config key: prefix_decay`: Old scripts passed `--prefix_decay`/`--prefix_alpha` which were removed when prefix_loss was rewritten. All 3 submission scripts (`submit_bpe_ablations.sh`, `submit_bpe_prefix_ablations.sh`, `submit_prefix_ablations.sh`) now fixed to use `--prefix_weight`. - Shell `source` error: SLURM `--wrap` uses `/bin/sh` which doesn't have `source`. Fixed to use POSIX `. "$CONDA_INIT"`. - Wrong BPE tokenizer: Default path had a 512-vocab test tokenizer. Trained new 32K BPE tokenizer. - Missing parquet data: C4 data was in `~/.cache` not in `base_data/`. Fixed with symlinks.

---

10. Infrastructure & Data

10.1 Compute

Resource	Details
Partition	`preemptible` (jobs preempted at ~2h wall time)
GPUs per job	2 (distributed via torchrun)

Resource	Details
Memory	64 GB
CPUs	8
Time limit	24 hours (but preempted much earlier)

10.2 Data Paths

Asset	Path	Size
LZ78 32K tokenizer	`.../lz78_ablations/tokenizers/lz78_32k/`	6 files
FreqGated 32K tokenizer	`.../lz78_ablations/tokenizers/freqgated_32k/`	6 files
Trie2x 44K tokenizer	`.../lz78_ablations/tokenizers/trie2x_44k/`	7 files (+ hier metadata)
BPE 32K tokenizer	`.../nanochat/tokenizer-32k/`	4 files
LZ78 pre-tokenized data	`.../lz78_ablations/data/lz78_32k/`	1.1 GB
FreqGated pre-tokenized data	`.../lz78_ablations/data/freqgated_32k/`	1.1 GB
Trie2x pre-tokenized data	`.../lz78_ablations/data/trie2x_44k/`	986 MB
C4 parquet data (BPE)	`.../nanochat/base_data/`	32 shards, ~2.9 GB

10.3 Each Tokenizer Directory Contains

File	Purpose
`lz78_codes.tsv`	Token dictionary (code, parent, char)
`lz78_config.json`	Tokenizer config (type, vocab size)
`token_bytes.pt`	Byte representation of each token (for BPB calc)
`token_metadata.pt`	(parent_code, char_byte) per token (for structured/tuple embedding)
`token_ancestors.pt`	(V, max_depth) ancestor chain indices (for prefix loss)
`token_ancestor_depths.pt`	(V,) depth per token (for prefix loss)
`token_metadata_hier.pt`	Trie parent metadata (Trie2x only, for hierarchical embedding)

---

11. Key Takeaways (So Far)

1. **BPE with standard CE is the best result: 0.9433 BPB.** BPE unchunked (0.9434) confirms this is not a fluke. BPE's advantage is in the tokenization algorithm itself.

2. **Chunking nearly halves the gap with BPE — the biggest finding.** Converged results: chunked FreqGated (1.0999), LZ78 (1.1016), Trie2x (1.1035) all reach ~1.10 BPB. Unchunked FreqGated (1.1756) is 24.6% behind BPE; chunked FreqGated is only 16.6% behind — cutting the gap from 24.6% to 16.6%. The three chunked tokenizers converge to nearly identical performance, suggesting regex pre-splitting standardizes the input.

3. **Prefix smoothing hurts ALL tokenizers — it is a negative result.** Even mild prefix smoothing (pw=0.1 → 1.0093, +7.0%) degrades BPE. The pattern is universal across all tokenizers.

4. **FreqGated LZ78 is the best LZ78-family tokenizer** — 1.1756 BPB unchunked (best), 1.0999 chunked (best). At convergence, FreqGated-flat (1.1756) beats LZ78-flat (1.1952) and Trie2x-hier (1.2306).

5. **Structured embedding catches up at convergence.** A surprise: LZ78-struct (1.1944) slightly beats LZ78-flat (1.1952), and FreqGated-struct (1.1768) nearly matches FreqGated-flat (1.1756). The structured advantage was invisible at step 2000 but emerges with more training. Tuple embedding also performs well: FreqGated-tuple (1.1762) matches flat.

6. **BPE's dominance is partly explained by better compression** (4.53 vs 3.90 bytes/token, +16%), but the quality gap (0.9433 vs 1.1756 = 24.6%) far exceeds this. Chunking closes a significant portion of the gap.

7. **Nearly all runs converged** to full Chinchilla-20 training. Only trie2x-44k-flat still running. 24 experiments total across 4 tokenizers, 4 embedding modes, 4 loss functions, and chunking ablation.

---

12. Next Steps

- ~~BPE standard baseline~~ **DONE** — 0.9433 BPB
- ~~Chunking ablation experiments~~ **DONE** — All 3 chunked runs converged: FG 1.0999, LZ78 1.1016, T2x 1.1035
- ~~BPE unchunked control~~ **DONE** — 0.9434 BPB, confirms BPE baseline (chunking not responsible for BPE advantage)
- ~~Full-convergence runs~~ **DONE** — 8 of 9 LZ78-family runs converged. Only trie2x-44k-flat still running (job 1706666)
- ~~Prefix smoothing experiments~~ **DONE** — Negative result confirmed across all tokenizers
- **Remaining**: trie2x-44k-flat full convergence (running), then all experiments complete
- **Future directions**: larger vocab sizes, chunking + structured embedding, different regex patterns, LZ78 with BPE-style merges