CONSTRUCTIVE PREDICTION: THE ROLE OF LEXICAL CODE ORGANIZATION IN LANGUAGE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Character, byte, and subword language models build text from categorical events that are individually valid but may combine into strings with no corresponding entry in a given lexicon. Thus, the prediction interface does not explicitly distinguish such constructions from those that complete a lexical entry. We introduce *constructive prediction*, which instead represents each lexical entry by a short path over a shared, low-cardinality alphabet. In this case, partial paths are intermediate shared states with many possible continuations, and only a completed path can either resolve to a lexical identity or remain unassigned. We instantiate this representation as Intuitionistic Language Models (ILM). We first test Flat ILM, which applies a causal Transformer to the coordinate stream. We then extend this design with Full ILM, which adds coordinate-role interfaces and prefix-aware supervision. To assess the effect of constructive prediction, our central experiment asks whether a lexical-to-path assignment constructed from external pretrained embeddings and frozen before downstream language-model training carries predictive structure. To do so, we reassign entries among the same set of paths under the Flat model. Using three assignment seeds, each evaluated with three training seeds, unrestricted reassignment worsens test BPB by 0.1442 ± 0.0014 on Tiny Shakespeare and 0.1817 ± 0.0093 on enwik8. All nine paired comparisons favor the original assignment on each corpus. To control for permutation-induced frequency shifts, we use a stricter control that shuffles only entries with equal training frequency, thereby preserving training-weighted coordinate frequencies at each path position. Reassignment still worsens BPB by 0.1063 ± 0.0079 and 0.0957 ± 0.0015, respectively. Beyond assignment, we ask whether the Transformer can exploit path structure. We find that Full ILM has lower mean BPB than Flat ILM at 6.5M and 15.5M on both corpora, and at 100M on enwik8. Full's advantage remains substantial on enwik8 but narrows to 0.0055 BPB at 15.5M on Tiny Shakespeare. Overall, these results show that the organization of lexical identities over shared sequential codes is a modeling variable that can improve prediction under a compact representation.

1 INTRODUCTION

An autoregressive language model builds text one categorical event at a time. Character and byte models use small alphabets, whereas subword tokenizers trade larger vocabularies for shorter sequences (Sennrich et al., 2016; Kudo & Richardson, 2018). Although each predicted event belongs to the model vocabulary, several events may combine into a string that does not correspond to any entry in a particular lexicon. The model can nevertheless continue assigning probability to that construction because lexical validity is not represented explicitly at the prediction interface. This flexibility supports open-vocabulary generation, but it also leaves lexical identity implicit in the event stream.

We study a complementary representation in which lexical identity is itself constructed. In *constructive prediction*, each entry in a fixed lexicon is assigned a short path of shared coordinates. A partial path represents an intermediate state, while a complete path assigned to the lexicon identifies exactly one entry. Prediction therefore proceeds through a sequence of decisions whose completion determines the lexical identity that has been constructed. We evaluate held-out next-event likelihood on observed occupied paths, separating predictive probability from the decoding rule. Free-running decoding would additionally require an occupied-path constraint to restrict generation to assigned paths.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We call models built on this representation Intuitionistic Language Models (ILM). The name draws a structural analogy with Brouwer's intuitionism, in which objects are given through construction grounded in temporal succession (van Atten, 2023). The analogy concerns the order between construction and identity. Specifically, in ILM, lexical identity is not treated as an indivisible outcome given independently of its construction, but is reached through an ordered sequence of coordinate decisions.

This construction connects to prior work on sequential lexical factorization. LightRNN learns a row-and-column code for each word (Li et al., 2016), whereas Hybrid-LightRNN reserves one-step codes for frequent words and learns two-step codes for the rest (Kong et al., 2019). Both optimize these assignments against the downstream language-model objective within methods primarily designed to reduce memory and computation. We instead freeze a fixed-length, low-cardinality map derived from external pretrained embeddings before downstream training and test whether its shared-prefix organization affects held-out prediction.

To test this question, we evaluate a closed-corpus instantiation whose deterministic scanner fixes the lexical vocabulary from the complete frozen corpus. Residual clustering of external pretrained embeddings and deterministic collision repair assign its entries to three- or four-coordinate paths over a base-64 alphabet. Flat ILM models the resulting coordinate stream with a standard causal Transformer, while Full ILM adds coordinate-role interfaces and prefix-aware supervision.

Our principal control isolates the lexical-to-code assignment by reassigning entries among the same set of paths while preserving the Flat model and training protocol. Three unrestricted maps freely alter the association, while three exact-frequency maps control for the resulting frequency shifts by shuffling only entries with equal training frequency, thereby preserving training-weighted coordinate frequencies at every path position. We provide two additional controls for alternative explanations based on multi-step lexical representation and optimization regime. First, Atomic Lexical preserves the same segmentation but collapses each lexical path to a single prediction event, thereby removing coordinate factorization. Second, an optimizer crossover compares Flat with the corresponding character or byte baseline before and after exchanging their optimization settings. All controls are first evaluated against Flat ILM at 6.5M, establishing a common compact regime. We separately examine capacity by comparing Flat and Full at 6.5M and 15.5M on both corpora and extending enwik8 to 100M. As a complementary higher-capacity check, we repeat Atomic at 15.5M on Tiny Shakespeare. Source contexts are approximately matched, and each comparison uses a common fixed training horizon.

Both assignment controls worsen held-out performance, with all nine paired comparisons favoring the original assignment on both corpora. Atomic has higher BPB than Flat at both Tiny Shakespeare tiers, while the optimizer crossover favors Flat over the corresponding character or byte baseline before and after exchanging their optimization settings. With the representation fixed, Full has lower mean BPB than Flat at 6.5M and 15.5M on both corpora and at 100M on enwik8. Its advantage narrows to 0.0055 BPB at 15.5M on Tiny Shakespeare but remains near 0.1 BPB across all three enwik8 tiers. Together, these results show that the lexical-to-code map is not a neutral relabeling and that the organization of lexical identities over shared sequential codes is itself a modeling variable.

2 Related Work

Tokenization and fine-grained language models. BPE and SentencePiece construct reusable subword vocabularies (Sennrich et al., 2016; Kudo & Richardson, 2018). Byte-level models rely on a fixed literal alphabet, while MEGABYTE, SpaceByte, and the Byte Latent Transformer add multiscale computation above that alphabet (Xue et al., 2022; Yu et al., 2023; Slagle, 2024; Pagnoni et al., 2024). BPE and SentencePiece jointly determine source segmentation and predicted events. ILM instead factorizes entries from a fixed lexical scanner. Its principal control preserves that segmentation and the occupied paths, varying only their lexical association.

Factorized lexical representations. LightRNN, Hybrid-LightRNN, KD codes, deep compositional code learning, and WEST represent lexical identities through shared components (Li et al., 2016; Kong et al., 2019; Chen et al., 2018; Shu & Nakayama, 2017; Variani et al., 2018). Related work also studies semantic and morphological parameterizations of token representations (Kavin R V &

Goyal, 2025; Reif et al., 2026) and fixed binary token codes (Bochkov, 2026), while hierarchical and adaptive softmaxes factor the output computation (Morin & Bengio, 2005; Grave et al., 2017).

LightRNN and Hybrid-LightRNN are the closest precedents for ILM because they likewise predict lexical-code components successively. Both learn their maps through the downstream language-model objective, and Hybrid-LightRNN compares learned, frequency-based, and random maps (Kong et al., 2019, Table 7). In contrast, ILM freezes a fixed-length map derived from external pretrained embeddings before downstream training. Replicated reassignments test whether this organization carries predictive structure while the occupied paths and Flat setup remain fixed. The exact-frequency control additionally preserves training-weighted coordinate frequencies.

Discrete embedding codes. The coordinate construction draws on product quantization and vector quantization (Jégou et al., 2011; van den Oord et al., 2017). Generative recommenders similarly decode embedding-derived semantic-ID sequences for items and compare them with random codes (Rajput et al., 2023). ILM instead tests lexical next-event prediction by reassigning the same occupied paths, with exact-frequency maps also fixing path and prefix counts.

3 Constructive Representation

A deterministic lexical scanner first turns text into a sequence of entries such as word-like spans, punctuation, whitespace, and line breaks. Let $\mathcal{W}$ denote the resulting vocabulary, and let $\mathcal{A}_K = \{0, \ldots, K-1\}$ be a shared alphabet. We represent each entry by an ordered length-S tuple over this alphabet through an injective map

$$\phi : \mathcal{W} \hookrightarrow \mathcal{A}_K^S, \qquad \phi(w) = (c_0(w), \ldots, c_{S-1}(w)). \tag{1}$$

For each $w \in \mathcal{W}$, we call $\phi(w)$ its *path* and $c_p(w)$ its *coordinate* at position p.

Distinct lexical entries therefore receive distinct paths. We call $\mathrm{im}(\phi)$ the *occupied path set*. Although K^S paths are possible, prediction proceeds through S successive choices among K coordinate values rather than one categorical choice over the lexical vocabulary. Relative to predicting the same lexical entries atomically, this representation uses S times as many model events and requires S coordinate decisions per generated entry. In our experiments, we use $K = 64$, with three coordinates per entry for Tiny Shakespeare and four for enwik8. Partial paths represent intermediate states, while a complete occupied path identifies exactly one lexical entry.

For the evaluated corpora, the scanner is lossless on enwik8, and its Tiny Shakespeare held-out encodings reconstruct their source ranges exactly. With the vocabulary fixed, we construct the paths by embedding each lexical entry with OpenAI `text-embedding-3-small` (OpenAI, 2024). After featurewise standardization and PCA, each entry is represented by a ten-dimensional vector. Successive spherical k-means stages then assign coordinates recursively. Specifically, the first stage clusters the embedding vectors, while later stages cluster residuals that result from removing the preceding centroid. These later residual centroids are then mapped back to the same 64-value alphabet by comparing them with the first-stage centroids. When preliminary paths collide, deterministic repair preserves the first assignment and reassigns later entries to unoccupied paths. It changes 20.4% of Tiny Shakespeare assignments and 32.5% of enwik8 assignments before producing the injective map ϕ. Appendix A gives the complete scanner, coordinate-construction, repair algorithm, and diagnostics. The map is constructed once before language-model training and remains fixed while the Transformer parameters are learned.

Once the map is fixed, the paths of lexical entries $(w_0, \ldots, w_{N-1})$ are flattened into a single coordinate stream,

$$x_{Sn+p} = c_p(w_n), \qquad 0 \le n < N, \quad 0 \le p < S. \tag{2}$$

Each x_t is a *coordinate event*. The index p is its *coordinate role*, which records the position of the event within a lexical path. Because every path has fixed length, this role is determined by the stream position modulo S. Earlier coordinates of the same path are therefore available as causal context when later coordinates are predicted.

For example, the frozen Tiny Shakespeare representation maps

$$\phi(\text{“The”}) = (2, 54, 30), \qquad \phi(\text{“ king”}) = (50, 54, 10),$$

so “The king” becomes the coordinate stream $(2, 54, 30, 50, 54, 10)$. Predicting 30 completes the first lexical path, while predicting 50 begins the next one. Writing $x_{<q}$ for all coordinate events preceding stream position q, the model assigns the lexical path probability

$$P_\theta(\phi(w_n) \mid x_{<Sn}) = \prod_{p=0}^{S-1} P_\theta(c_p(w_n) \mid x_{<Sn+p}). \tag{3}$$

Each factor is a K-way softmax probability, with $K = 64$ in our experiments. Later coordinates can therefore use earlier coordinates from the same lexical path as causal context, while the first coordinate is predicted only from the preceding text. In the example, 2 is available when predicting 54, and $(2, 54)$ is available when predicting 30. Appendix B contrasts this factorized path probability with direct prediction through a vocabulary-sized lexical softmax.

4 MODELS

Flat ILM. Flat ILM is the direct baseline. It treats the coordinate stream in Equation 2 exactly as a standard decoder-only Transformer treats a token stream. With hidden width C, coordinate embedding table $U_{\text{coord}} \in \mathbb{R}^{K\times C}$, and learned absolute positional embedding $\text{PE}_t \in \mathbb{R}^C$, the input at position t is

$$H_t^{(0)} = U_{\text{coord}}[x_t] + \text{PE}_t. \tag{4}$$

Standard causal self-attention and feed-forward blocks then produce a shared K-way classifier for the next coordinate. Flat therefore changes what each autoregressive event represents while retaining ordinary Transformer computation. It receives no explicit path-boundary marker or coordinate-role label.

Full ILM. Full ILM retains the same coordinate paths and Transformer backbone while modifying the input, output, and training interfaces. It uses a separate embedding table for each coordinate role and a separate next-coordinate classifier for each target role. It also drops loss terms whose context states belong to an incomplete path fragment at the left edge of a sampled window. Supervision resumes once the context reaches a path whose prefix begins inside the window. These changes are enabled together, so the Flat-to-Full comparison measures their combined effect. The additional interfaces make Full 0.8–1.8% larger than Flat at the 6.5M and 15.5M tiers. Appendix C.2, Table 7, gives the exact counts.

For each role $p \in \{0, \ldots, S-1\}$, Full uses input table $U_p \in \mathbb{R}^{K\times C}$, output matrix $W_p \in \mathbb{R}^{C\times K}$, and bias $\beta_p \in \mathbb{R}^K$. For batch element b, let r_b denote the coordinate role of the first event in its sampled window. At coordinate-time position t, the input role is $(r_b + t) \bmod S$ and the next-coordinate role is $(r_b + t + 1) \bmod S$. These roles depend only on stream position and do not reveal the coordinate value being predicted. Let $H_{b,t}^{(L)} \in \mathbb{R}^C$ denote the state after L Transformer blocks and $Z_{b,t} \in \mathbb{R}^K$ the next-coordinate logits. Then

$$H_{b,t}^{(0)} = U_{(r_b+t) \bmod S}[x_{b,t}] + \text{PE}_t, \tag{5}$$

$$Z_{b,t} = H_{b,t}^{(L)} W_{(r_b+t+1) \bmod S} + \beta_{(r_b+t+1) \bmod S}. \tag{6}$$

The target is $y_{b,t} = x_{b,t+1}$. Let $m_{b,t}^{\text{prefix}} \in \{0, 1\}$ equal zero when context position t belongs to the incomplete left-edge path fragment and one otherwise. Full ILM uses the masked objective

$$\mathcal{L}_{\text{prefix}} = -\frac{\sum_{b,t} m_{b,t}^{\text{prefix}} \log \text{softmax}(Z_{b,t})_{y_{b,t}}}{\sum_{b,t} m_{b,t}^{\text{prefix}}}. \tag{7}$$

The mask therefore excludes predictions made from the incomplete path fragment and retains later predictions once the context state’s within-path prefix is fully available in the window. It changes only which training losses are included. Causal attention and coordinate-by-coordinate inference otherwise retain their usual form. Appendix B gives the complete role and masking formalism.

Controls. The controls separate coordinate factorization from the lexical-to-code assignment. First, Atomic Lexical keeps the same segmentation but replaces each length-S coordinate path with one categorical event for the corresponding lexical entry, providing a segmentation-preserving comparison without coordinate factorization. Its vocabulary-sized input and output tables require a narrower Transformer to match total parameter count, so the control is parameter-matched rather than backbone-matched.

Second, Permuted Flat retains the Flat architecture and changes only the assignment of lexical entries to occupied paths. For assignment seed r, let $\pi_{\mathcal{C},r}$ be a sampled bijection of the occupied path set. The reassigned map is

$$\phi_{\pi_r} = \pi_{\mathcal{C},r} \circ \phi, \qquad \pi_{\mathcal{C},r} : \mathrm{im}(\phi) \to \mathrm{im}(\phi). \tag{8}$$

Segmentation, occupied paths, alphabet, path length, architecture, and optimization therefore remain unchanged. Only the lexical entry associated with each path changes. The exact-frequency variant further restricts reassignment to entries that occur equally often in the training split, thereby preserving training-weighted coordinate frequencies at each path position.

5 Experimental Design

Corpora and frozen representations. We evaluate Tiny Shakespeare (Karpathy, 2015) and enwik8 (Hutter, 2006). Tiny Shakespeare uses ordered, disjoint training, validation, and test regions separated by boundary gaps, while enwik8 uses the canonical 90M/5M/5M byte partition. We separate representation construction from model fitting. For each corpus, the scanner is fixed, and the vocabulary and final collision-repaired lexical-to-code map are constructed once from the complete frozen corpus. Tiny Shakespeare uses paths of length $S = 3$, while the larger enwik8 vocabulary uses $S = 4$. The experiments therefore study closed-corpus representations whose lexical inventories are known before model training, rather than tokenizers learned exclusively from the training split. Once constructed, the representation remains fixed while each Transformer is trained on the training split. Validation data monitor optimization, and test data are reserved for the final BPB comparisons described below. Appendix A specifies the representation construction and reconstruction checks, while Appendix C.1 records the splits and evaluation rules.

Questions and controls. Our principal control experiment asks whether the assignment created in Section 3 matters after the occupied path set and the Flat model have been fixed. The PCA and clustering stages construct this assignment jointly across the vocabulary. The permutation controls reassign entries among the fixed occupied paths, changing which entries share intermediate prefixes. Unrestricted Permuted Flat permits any such reassignment. Because lexical entries occur with different frequencies, this can also change coordinate frequencies. The exact-frequency control instead shuffles only entries that occur equally often in the training split. Consequently, every path retains its training count, preserving all within-path prefix counts and conditional next-coordinate frequencies. Assignment seeds 314159, 271828, and 161803 sample three maps. Training seeds 13, 29, and 47 vary model initialization and stochastic training order for each map, giving nine paired comparisons with the original Flat assignment on each corpus.

Supporting comparisons address alternative explanations. Atomic Lexical retains the scanner but predicts each lexical entry as a single categorical event, providing a segmentation-preserving comparison without coordinate factorization. Character GPT and Byte GPT provide direct raw-symbol references. We implement them in nanoGPT (Karpathy, 2022), a transparent and widely used decoder-only implementation with an established Tiny Shakespeare setup that we configure at the same 6.5M parameter tier. The byte-level configuration provides the analogous enwik8 reference without requiring a learned tokenizer. Finally, Flat and Full share the same lexical-to-code map, so their comparison tests role-specific interfaces and prefix-aware training together.

Parameter budgets and source context. We organize the experiments into three parameter tiers. At 6.5M, the compact scale makes it tractable to evaluate every assignment seed with every training seed, so this tier contains all principal controls. At 15.5M, we repeat Atomic Lexical on Tiny Shakespeare and compare Flat with Full on both corpora. The 100M enwik8 tier then tests whether the Flat-to-Full ordering persists at greater capacity. Appendix C.2, Table 7, gives the complete experiment matrix, widths, and exact trainable parameter counts.

Atomic Lexical requires separate parameter matching because its vocabulary-sized input and output interfaces consume more parameters than the coordinate interfaces of Flat and Full. We therefore evaluate it at two hidden widths. At width 156, its total parameter count matches the 6.5M tier, while width 300 places it at the 15.5M tier. This allows Atomic Lexical to be compared with the coordinate models at both parameter scales despite their different interface costs. The comparison is practical on Tiny Shakespeare. On enwik8, however, the much larger vocabulary would cause the atomic interfaces to dominate either parameter tier.

Parameter matching alone does not ensure that models receive comparable amounts of source context. We therefore use coordinate windows spanning 20 lexical-path equivalents. Their 60 events on Tiny Shakespeare and 80 events on enwik8 cover 73.9 and 72.2 represented training-source bytes on average, respectively. The raw-symbol references use 74 events and cover 74 source bytes, closely matching source span but not event count or attention computation. This difference applies only to those supporting comparisons. Original and Permuted Flat instead use identical event counts and share the same architecture, context, and training conditions. To further align the compact comparisons, all 6.5M and 15.5M models share six blocks, six heads, the same training seeds, and a 6,000-update horizon. For the 100M extension, Flat and Full retain the enwik8 representation, context, training seeds, depth, and horizon while increasing model width.

Optimization protocols. Under their native settings, the nanoGPT reference uses warmup and decay, whereas the ILM protocol uses a constant learning rate. A direct comparison would therefore vary both representation and optimization, so the observed BPB difference could reflect either. To separate these effects, at 6.5M we train each raw-symbol reference and its corresponding Flat ILM model under both optimization protocols. This 2×2 crossover tests whether the Flat-versus-reference BPB ordering persists when the optimization settings are exchanged. Because the optimization protocol may affect the magnitude of the gap, we use consistency of its direction across both protocols as the relevant supporting evidence rather than relying on a single cross-implementation BPB difference. Appendix C.2, Table 8, specifies both protocols. Outside the crossover, all 6.5M and 15.5M runs use the constant-rate ILM protocol.

At 100M, we use a separate large-scale regime in which Flat and Full share the same AdamW cosine schedule. Appendix C.4 specifies this protocol and the pilot that motivated it.

Likelihood metric and evaluation. Across Atomic Lexical, Flat, Full, and the raw-symbol references, the same source text is represented by different numbers of prediction events. Their per-event losses are therefore not directly comparable. To place these models on a common source-level scale, we report teacher-forced bits per UTF-8 source byte (BPB), for which a lower value is better. BPB sums next-event prediction losses and normalizes them by the amount of represented source text. Under teacher forcing, each target is scored from its observed held-out context rather than from previously generated events.

Because the two corpora differ substantially in test-set size, we use two teacher-forced evaluation procedures. Tiny Shakespeare uses rolling full-context evaluation, while enwik8 uses block-reset evaluation to score its five-million-byte test region tractably. Every model family within a corpus follows the same evaluation rule. Full's prefix-aware mask applies only during training, so all reported test BPB values use ordinary next-event losses. Appendix C.1 gives the BPB formula, event-to-byte weighting, worked examples, both context rules, and the calculation of the exact scored-byte denominator.

Unless stated otherwise, model-family means and sample standard deviations summarize the three training seeds, and paired gaps use matching training seeds. Permutation aggregates instead summarize three assignment-level means, each averaged over the same three training seeds.

6 Results

6.1 The lexical-to-code assignment matters

We begin with the main assignment control at 6.5M. First, the unrestricted control reassigns lexical entries among the same occupied paths while holding the Flat setup fixed. If the association between entries and paths aids prediction, BPB should worsen. We generate three unrestricted maps using

three assignment seeds, and evaluate each map with training seeds 13, 29, and 47, yielding nine paired comparisons per corpus.

Across the three assignment-level means, unrestricted reassignment worsens BPB by 0.1442 ± 0.0014 on Tiny Shakespeare and 0.1817 ± 0.0093 on enwik8. All nine paired BPB differences are positive on both corpora, as shown in Figure 1 and the complete matrix in Appendix D.2, Table 12.

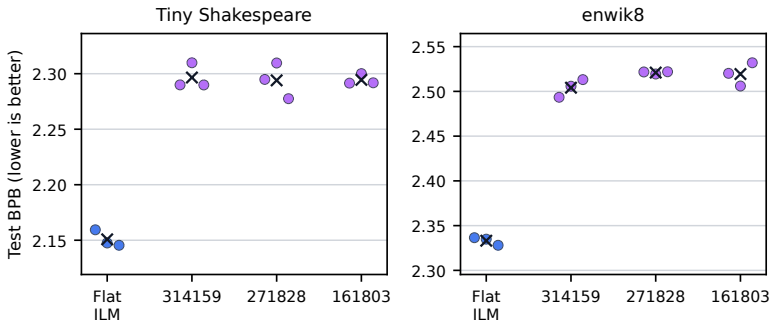


Figure 1: Replicated permutation control at approximately 6.5M parameters. Round markers show training-seed results. Black $\times$ markers are means over training seeds 13, 29, and 47 within a fixed assignment. The vertical axes are separately scaled by corpus.

Because unrestricted reassignment preserves the occupied paths and Flat-model conditions, its penalty isolates the lexical-to-code association rather than code capacity or factorization. However, it can also change how often coordinate values appear at each path position during training when frequent entries receive different paths. Its BPB penalty could therefore partly reflect this frequency shift. To address this, the exact-frequency control shuffles only entries that occur equally often in the training data. BPB still worsens in all nine paired comparisons on each corpus, with mean penalties of 0.1063 ± 0.0079 BPB on Tiny Shakespeare and 0.0957 ± 0.0015 BPB on enwik8 (Appendix D.3, Table 13).

The exact-frequency maps move 98.1% of Tiny Shakespeare's entries and 99.6% of enwik8's entries on average. However, these percentages count entries equally. To account for how often each entry actually appears, we also report *moved occurrence mass*, which weights entries by their occurrences in the training dataset. By this measure, reassigned entries account for 31.3% of the Tiny Shakespeare training stream and 23.5% of the enwik8 training stream (Appendix D.1).

6.2 Optimization robustness against raw-symbol references

Having established that the assignment matters within Flat, we next compare Flat with familiar raw-symbol GPT references. Using nanoGPT, we instantiate Character GPT on Tiny Shakespeare's 65-character alphabet, while on enwik8 we instantiate Byte GPT on the 205 byte values observed in the frozen corpus. However, nanoGPT and ILM were originally trained with different optimization protocols, so a native-only comparison would change both representation and optimization at once. To separate these effects, we evaluate each reference and its matching Flat model before and after exchanging their optimization protocols.

Table 1: Optimizer crossover at approximately 6.5M parameters. Each cell is mean BPB $\pm$ sample standard deviation over training seeds 13, 29, and 47. Lower is better.

	Optimizer protocol			
	ILM AdamW		nanoGPT AdamW	
Corpus	Reference GPT	Flat ILM	Reference GPT	Flat ILM
Tiny Shakespeare	2.6506 ± 0.0373	2.1508 ± 0.0075	2.7167 ± 0.0168	2.1220 ± 0.0044
enwik8	2.4797 ± 0.0160	2.3331 ± 0.0045	2.6401 ± 0.0039	2.3082 ± 0.0019

Table 1 shows that optimization affects both absolute BPB and the size of the gap. The raw-symbol references improve under the ILM protocol, whereas Flat improves under nanoGPT. Nevertheless, Flat remains lower on both corpora, under both protocols, and for every shared training seed. The mean paired reference-minus-Flat differences are 0.4998 ± 0.0365 and 0.1465 ± 0.0122 BPB under the ILM protocol for Tiny Shakespeare and enwik8, respectively. Under nanoGPT, they are 0.5947 ± 0.0124 and 0.3319 ± 0.0056 BPB. Thus, the protocol changes the size of the gap but not its direction. Subsequent comparisons at 6.5M and 15.5M use the ILM protocol.

6.3 Full ILM across the 6.5M and 15.5M tiers

We then ask whether, with the lexical-to-code assignment fixed, Full improves on Flat by explicitly modeling the position of each coordinate within a lexical path. Full combines role-specific interfaces with prefix-aware supervision. We compare the models at 6.5M and 15.5M on both corpora, then extend enwik8 to 100M in Section 6.4.

Table 2: Teacher-forced BPB for the ILM-family models at 6.5M and 15.5M parameters. Values are mean $\pm$ sample standard deviation over three training seeds. Lower is better.

	Tiny Shakespeare		enwik8	
Family	6.5M	15.5M	6.5M	15.5M
Atomic Lexical	2.3205 ± 0.0240	2.3840 ± 0.0061	–	–
Flat ILM	2.1508 ± 0.0075	2.1406 ± 0.0098	2.3331 ± 0.0045	2.2928 ± 0.0095
Full ILM	2.1201 ± 0.0086	2.1351 ± 0.0021	2.2359 ± 0.0063	2.1815 ± 0.0020

6.5M tier. First, Full ILM improves on Flat at 6.5M under the shared ILM optimization protocol. The mean paired Flat-minus-Full difference is 0.0307 ± 0.0086 BPB on Tiny Shakespeare and 0.0972 ± 0.0041 BPB on enwik8.

Separately, as a parameter-matched supporting comparison, Atomic Lexical preserves the same segmentation while removing coordinate factorization by predicting each lexical entry as a single event. It reaches 2.3205 ± 0.0240 BPB, compared with Flat's 2.1508 ± 0.0075. Appendix C.2, Table 7, gives its narrower configuration, while Appendix C.3, Figure 5, shows the training-seed values and repeated Flat-to-Full ordering.

15.5M tier. We then repeat these comparisons at 15.5M. Full's mean remains lower on both corpora under the same training protocol, but its advantage differs by corpus. On enwik8, the mean paired Flat-minus-Full difference is 0.1112 ± 0.0078 BPB. From 6.5M to 15.5M, Flat improves from 2.3331 ± 0.0045 to 2.2928 ± 0.0095 and Full from 2.2359 ± 0.0063 to 2.1815 ± 0.0020. On Tiny Shakespeare, the paired difference instead shrinks to 0.0055 ± 0.0094 BPB over the common 6,000-update horizon.

At 15.5M, Atomic also remains above Flat on Tiny Shakespeare, reaching 2.3840 ± 0.0061 BPB compared with Flat's 2.1406 ± 0.0098. Thus, under the tested total-parameter allocations, the shared lexical segmentation does not reproduce Flat's lower BPB. As discussed in the experimental design, Atomic Lexical is evaluated only on Tiny Shakespeare because its much larger enwik8 vocabulary would cause the atomic interfaces to dominate the parameter budget.

6.4 Capacity behavior and the 100M enwik8 extension

The 15.5M results reveal a different pattern across the two corpora. Full's advantage nearly vanishes on Tiny Shakespeare but remains substantial on enwik8. To test whether the enwik8 advantage persists at greater capacity, we extend that comparison to 100M parameters. At width 1,176, Flat has 99,891,856 trainable parameters and Full has 100,343,632, a 0.45% difference arising from Full's role-specific interfaces.

A constant-rate pilot showed late-stage instability, so the 100M Flat and Full models use a shared cosine schedule (Appendix C.4). Under this schedule, Flat obtains 2.2100 ± 0.0021 BPB and Full improves on it by 0.0966 ± 0.0015 BPB, reaching 2.1134 ± 0.0016. Full is lower for all three training

seeds (Figure 2). At the common 6,000-update horizon, the 100M result therefore measures whether this ordering persists under the larger-capacity cosine regime.

Appendix C.5 provides context for the absolute BPB values using results obtained under different protocols.

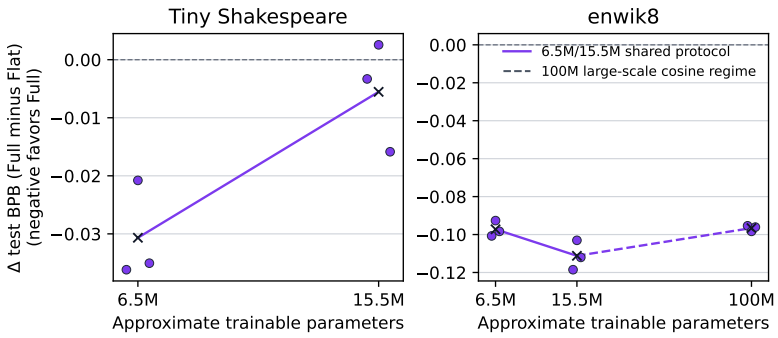


Figure 2: Full-minus-Flat BPB diagnostic across completed parameter tiers. Negative values favor Full ILM. Solid segments connect the 6.5M and 15.5M tiers under the shared constant-rate ILM protocol. The dashed enwik8 segment denotes the 100M cosine regime and cross-regime persistence (Appendix C.4). Circles show training-seed results, and black $\times$ markers show means.

7 Discussion

The experiments identify two levels at which constructive representation affects predictive likelihood. First, the lexical-to-code assignment itself matters. Reassignment worsens held-out likelihood even when exact-frequency maps preserve path and prefix counts. The penalty therefore depends on lexical-to-path association beyond within-path frequencies, including which entries share intermediate prefixes. The within-Flat permutations show that the final frozen map, which combines embedding-derived clustering with deterministic collision repair, carries predictive structure as a whole. Atomic Lexical and the raw-symbol references provide supporting comparisons because they also change the representation and its access to the fixed embedding prior. Under the tested total-parameter allocations, preserving the same segmentation in Atomic Lexical does not reproduce Flat's lower BPB. Flat also remains below the Character and Byte GPT references before and after exchanging optimization protocols.

Second, Full ILM explicitly models structure within each path through role-specific interfaces and prefix-aware supervision. Its mean BPB is lower than Flat's in every compact comparison and for all three 100M enwik8 runs. The benefit shrinks to 0.0055 BPB on Tiny Shakespeare at 15.5M while remaining near 0.1 BPB on enwik8 across all three tiers. This contrast motivates studies that vary corpus scale, data exposure, capacity, and optimization state, together with ablations of Full's two components. The compact comparisons approximately align parameter count, source span, and training horizon, while coordinate models process several events per lexical entry. Compute-matched, longer-context, open-vocabulary, and free-running studies could broaden the evidence beyond this held-out likelihood setting.

8 Conclusion

Constructive prediction represents lexical identity through shared coordinate decisions. Reassignment raises BPB even when training-weighted coordinate frequencies are preserved, while Atomic Lexical and the optimizer crossover provide supporting comparisons for segmentation and optimization. Full ILM has lower mean BPB in every completed compact corpus-tier comparison and lower BPB in all three 100M enwik8 training-seed runs. Thus, both the lexical-to-code map and the way its paths are modeled are consequential choices in autoregressive language modeling.

Reproducibility Statement

The appendix specifies the coordinate construction, likelihood and prefix formalism, data splits, model dimensions, optimization regimes, exact parameter counts, and evaluation rules. The supplementary anonymous code package includes commands and configurations for the 6.5M/15.5M, optimizer-crossover, permutation, and 100M extension runs, together with frozen tokenizer mappings, split manifests, assignment maps, results for every assignment seed and training seed, and artifact-generation scripts.

AI Use Statement

Generative AI tools were used for feedback on experimental framing and presentation, interpretation and presentation of results, literature positioning, code editing, and manuscript editing. They were also used to edit mathematical exposition for clarity. The mathematical formulation, model design, experimental methodology, and research ideas were developed by the authors. The authors independently conducted the experiments, verified the reported numerical results, implementation behavior, and citations, reviewed all AI-assisted content, and take responsibility for the complete manuscript.

References

Andrey Bochkov. Language models without a trainable input embedding table: Learning from fixed minimal binary token codes. *arXiv preprint arXiv:2605.09751*, 2026. URL https://arxiv.org/abs/2605.09751.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning K-way D-dimensional discrete codes for compact embedding representations. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 854–863. PMLR, 2018. URL https://proceedings.mlr.press/v80/chen18g.html.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1285. URL https://aclanthology.org/P19-1285/.

Édouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for GPUs. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1302–1310. PMLR, 2017. URL https://proceedings.mlr.press/v70/grave17a.html.

Marcus Hutter. The Hutter prize for human knowledge compression. https://www.hutter1.net/prize/, 2006. enwik8 benchmark corpus.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57. URL https://ieeexplore.ieee.org/document/5432202.

Robin-Nico Kampa, Fabian Deuser, Anna Boßendörfer, Konrad Habel, and Norbert Oswald. 1GC-7RC: One graphic card – seven research challenges! how good are AI agents at doing your job? *arXiv preprint arXiv:2605.17046*, 2026. URL https://arxiv.org/abs/2605.17046.

Andrej Karpathy. char-rnn. https://github.com/karpathy/char-rnn, 2015. Tiny Shakespeare source, accessed 2026-08-19.

Andrej Karpathy. nanoGPT: The simplest, fastest repository for training/finetuning medium-sized GPTs. https://github.com/karpathy/nanoGPT, 2022. GitHub repository.

Kavin R V and Pawan Goyal. Breaking token into concepts: Exploring extreme compression in token representation via compositional shared semantics. *arXiv preprint arXiv:2509.17737*, 2025. URL https://arxiv.org/abs/2509.17737.

Xiang Kong, Qizhe Xie, Zihang Dai, and Eduard Hovy. Fast and simple mixture of softmaxes with BPE and Hybrid-LightRNN for language generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6626–6633, 2019. doi: 10.1609/aaai.v33i01.33016626. URL https://arxiv.org/abs/1809.09296.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012/.

Xiang Li, Tao Qin, Jian Yang, and Tie-Yan Liu. LightRNN: Memory and computation-efficient recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016. URL https://arxiv.org/abs/1610.09893.

Frédéric Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In Robert G. Cowell and Zoubin Ghahramani (eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, volume R5 of *Proceedings of Machine Learning Research*, pp. 246–252. PMLR, 2005. URL https://proceedings.mlr.press/r5/morin05a.html.

OpenAI. text-embedding-3-small model. https://developers.openai.com/api/docs/models/text-embedding-3-small, 2024. Accessed 2026-08-19.

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

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10299–10315, 2023. doi: 10.52202/075280-0452. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/20dcab0f14046a5c6b02b61da9f13229-Abstract-Conference.html.

Yuval Reif, Guy Kaplan, and Roy Schwartz. Vocab diet: Reshaping the vocabulary of LLMs via vector arithmetic. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32334–32352. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1618. URL https://aclanthology.org/2026.findings-acl.1618/.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/.

Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*, 2017. URL https://arxiv.org/abs/1711.01068.

Kevin Slagle. SpaceByte: Towards deleting tokenization from large language modeling. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3967. URL https://arxiv.org/abs/2404.14408.

Roberto Tacconelli. StateSMix: Online lossless compression via Mamba state space models and sparse n-gram context mixing. *arXiv preprint arXiv:2605.02904*, 2026. URL https://arxiv.org/abs/2605.02904.

Mark van Atten. The development of intuitionistic logic. In Edward N. Zalta and Uri Nodelman (eds.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2023 edition, 2023. URL https://plato.stanford.edu/archives/fall2023/entries/intuitionistic-logic-development/.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://arxiv.org/abs/1711.00937.

Ehsan Variani, Ananda Theertha Suresh, and Mitchel Weintraub. WEST: Word encoded sequence transducers. *arXiv preprint arXiv:1811.08417*, 2018. URL https://arxiv.org/abs/1811.08417.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17/.

Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2305.07185.

A COORDINATE CONSTRUCTION

This section gives the complete construction of the lexical-to-code map used in the experiments. For each corpus, the scanner first determines a lexical vocabulary. Each entry is then embedded and successively clustered through residual stages. The resulting stage assignments are mapped into a shared coordinate alphabet, after which collision repair produces the final injective map ϕ. The representation is constructed once and then held fixed during language-model training.

A.1 LEXICAL SEGMENTATION AND SOURCE RECONSTRUCTION

The scanner applies a fixed, ordered grammar to the source text. It recognizes ASCII alphanumeric runs, punctuation from `.,:;()!?`, `+-_*`, and `&^%#={}'"`, explicit bracket and math delimiters, newline, carriage-return, and tab characters, and runs of at least two ASCII spaces. Form feeds and vertical tabs may prefix an alphanumeric run or a supported punctuation group.

Single spaces and longer runs are treated differently. For example, "The king" is segmented as "The" followed by " king", so the single space becomes part of the following lexical entry. By contrast, a source span consisting of "The", two spaces, and "king" is segmented as "The", a two-space entry, and "king". A single space can likewise remain attached to a following bracket, math delimiter, or supported punctuation group other than a period or comma.

For enwik8, the scanner operates losslessly. It retains the exact spelling of every match, emits each unmatched character as a one-character entry, and performs no case folding or Unicode normalization. Consequently, concatenating the emitted entries exactly reconstructs the source. Tiny Shakespeare uses the earlier scanner configuration, which strips attached spaces from a small set of explicit bracket and math delimiters and omits unmatched characters. Its validation and test encodings nevertheless reconstruct their source ranges exactly. The training encoding omits one isolated terminal space and therefore represents 892,314 of 892,315 source bytes. These reconstruction properties are checked before likelihood evaluation.

A.2 EMBEDDING INITIALIZATION AND RESIDUAL CLUSTERING

Once segmentation fixes the vocabulary $\mathcal{W}$, each lexical entry is represented by an embedding vector. The OpenAI `text-embedding-3-small` model (OpenAI, 2024) returns a 1,536-dimensional embedding for each entry. Newline, carriage-return, and tab entries are supplied as `<newline>`, `<carriage return>`, and `<tab>`. A whitespace-only entry is supplied as `<whitespace length=`n`>`, where n is its character length. Every other entry is stripped only for the embedding request, while its original spelling remains the lexical entry represented by the map. Distinct scanner entries can therefore produce identical embedding requests. They remain distinct vocabulary entries, and collision repair resolves any resulting preliminary path collision.

The complete vocabulary embedding matrix is standardized featurewise using scikit-learn's `StandardScaler`. Individual vectors are not L2-normalized at this stage. PCA is then fitted to the same matrix with ten requested components, `svd_solver=auto`, and random state 13. Both evaluated vocabularies retain all ten components. We write

$$E_0(w) \in \mathbb{R}^{10} \tag{9}$$

for the resulting vector associated with entry w. Ten dimensions are a fixed preprocessing choice rather than a value selected using held-out loss.

Starting from $E_0(w)$, the construction performs S residual clustering stages. At each stage, entries are grouped by the direction of their current residual. The selected cluster determines the stage assignment, while the mean residual of that cluster is removed before the next stage.

At stage p, define the normalized residual

$$\widehat{E}_p(w) = \frac{E_p(w)}{\max(\|E_p(w)\|, 10^{-12})}. \tag{10}$$

For each stage, scikit-learn KMeans on the normalized residuals provides the initial labels and centers using `n_init=auto` and random state $13 + p$. The centers are then L2-normalized before spherical

refinement. The stage-specific random states are fixed in advance rather than selected using held-out loss.

The stage uses two centers for each nonempty cluster because assignment and residual subtraction operate on different quantities. Let $a_p(w)$ denote the final cluster label of entry w, and let

$$\mathcal{W}_{p,j} = \{w \in \mathcal{W} : a_p(w) = j\}$$

collect the entries assigned to cluster j at stage p. The direction center and residual mean are

$$\begin{aligned} D_{p,j} &= \frac{\sum_{w \in \mathcal{W}_{p,j}} \widehat{E}_p(w)}{\max\left(\left\|\sum_{w \in \mathcal{W}_{p,j}} \widehat{E}_p(w)\right\|, 10^{-12}\right)}, \\ C_{p,j} &= \frac{1}{|\mathcal{W}_{p,j}|} \sum_{w \in \mathcal{W}_{p,j}} E_p(w). \end{aligned} \tag{11}$$

Both centers use the same cluster members. However, $D_{p,j}$ averages normalized residual directions, while $C_{p,j}$ averages the original residual vectors and therefore retains magnitude information. In general, $D_{p,j}$ is not the normalized version of $C_{p,j}$.

The direction center determines the cluster assignment, while the corresponding residual mean determines what is removed:

$$a_p(w) = \arg\max_{j \in \mathcal{A}_K} \widehat{E}_p(w)^\top D_{p,j}, \tag{12}$$

$$E_{p+1}(w) = E_p(w) - C_{p,a_p(w)}, \qquad 0 \le p < S - 1. \tag{13}$$

Spherical refinement alternates cosine-direction assignment with center recomputation. It stops when the labels are unchanged and the largest center shift is below 10^{-4}, or after 100 iterations. An empty cluster retains its previous direction center. Its residual mean falls back to that retained direction center, but no entry has the empty label, so the fallback is never subtracted from an entry. NumPy's `argmax` and `argsort` ordering resolve exact ties under the recorded software environment.

Figure 3 illustrates one residual stage. The two-dimensional schematic uses four clusters so that the geometry remains visible. The reported construction applies the same operations in ten dimensions with $K = 64$.

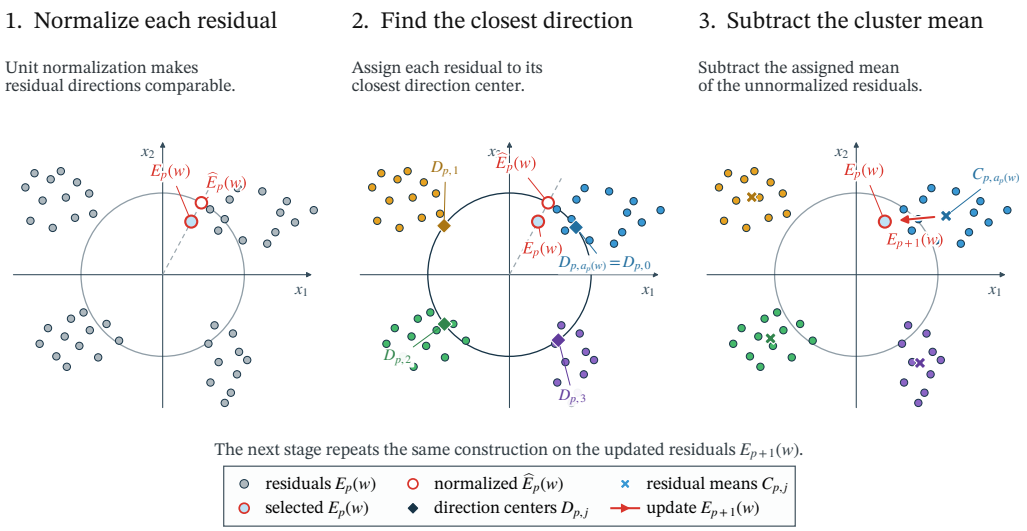


Figure 3: **One stage of residual coordinate construction.** Each residual is first normalized and assigned to the closest unit direction center $D_{p,j}$ by cosine similarity. The unnormalized members of the selected cluster then define the residual mean $C_{p,j}$, which is subtracted to obtain $E_{p+1}(w)$. Direction centers and residual means use the same cluster members, but they need not point in the same direction because normalization occurs before averaging only for $D_{p,j}$. Repeating the stage produces the residual-cluster assignments used to construct the coordinate path.

A.3 SHARED COORDINATE ALPHABET

The residual stages above are fitted independently, so their labels $a_p(w)$ are local to each stage. They cannot yet be interpreted as coordinates from one shared alphabet. We therefore map every stage into the common alphabet

$$\mathcal{A}_K = \{0, \dots, K-1\},$$

with $K = 64$, using the first-stage residual centroids as references.

At the first stage, the local labels already define the reference alphabet, so

$$\widetilde{c}_0(w) = a_0(w).$$

For each later stage $p \geq 1$, the construction compares the selected residual centroid $C_{p,a_p(w)}$ with the first-stage centroids $C_{0,i}$ and assigns the index with the closest direction:

$$\widetilde{c}_p(w) = \arg\max_{i \in \mathcal{A}_K} \frac{C_{p,a_p(w)}^\top C_{0,i}}{\max(\|C_{p,a_p(w)}\|\|C_{0,i}\|, 10^{-12})}. \tag{14}$$

This alignment is not constrained to be bijective, so distinct residual clusters at a later stage can map to the same shared coordinate before collision repair.

Thus, $a_p(w)$ records which residual cluster is selected at stage p, while $\widetilde{c}_p(w)$ records the corresponding value in the shared coordinate alphabet. Collecting these values defines the preliminary map

$$\widetilde{\phi} : \mathcal{W} \to \mathcal{A}_K^S, \qquad \widetilde{\phi}(w) = (\widetilde{c}_0(w), \dots, \widetilde{c}_{S-1}(w)). \tag{15}$$

Because different lexical entries can receive the same preliminary path, $\widetilde{\phi}$ need not be injective. The final step repairs these collisions.

A.4 COLLISION REPAIR AND DIAGNOSTICS

Collision repair transforms the preliminary map $\widetilde{\phi}$ into the final injective map

$$\phi : \mathcal{W} \hookrightarrow \mathcal{A}_K^S, \qquad \phi(w) = (c_0(w), \dots, c_{S-1}(w)). \tag{16}$$

If $\widetilde{\phi}(w)$ is unoccupied when entry w is processed, it is retained unchanged. Otherwise, repair searches for an unoccupied path while preserving the longest possible prefix of $\widetilde{\phi}(w)$. Entries are processed in first-appearance order, so the first entry assigned to a path retains it. The repair is therefore deterministic but order-dependent.

For a colliding entry, the 64 coordinate values at each remaining position are ranked by cosine similarity between that entry's stage residual and the first-stage reference centroids. Candidate completions are enumerated as Cartesian products of these per-position rankings, beginning with changes to the final coordinate and shortening the preserved prefix only when necessary. Within each product, the implementation follows the positional ranking lists in order, with the rightmost coordinate changing fastest. These rankings are precomputed from the original stage residuals and are not updated when a candidate prefix changes. The first unoccupied path is selected.

Because each position eventually considers all 64 coordinate values, the search can enumerate all 64^S paths. Since the construction checks $|\mathcal{W}| \leq 64^S$ before repair, an unoccupied path exists whenever a collision is processed. The procedure therefore terminates with an injective map for both evaluated vocabularies.

Tiny Shakespeare contains 15,030 lexical entries and uses $64^3 = 262{,}144$ possible paths. Collision repair changes 3,063 preliminary assignments, or 20.4%. enwik8 contains 519,955 lexical entries and uses $64^4 = 16{,}777{,}216$ possible paths. Collision repair changes 168,839 preliminary assignments, or 32.5%.

The final representation is injective but not surjective. Teacher-forced evaluation traverses only coordinate paths associated with held-out lexical entries. The current free-running decoder samples from the full coordinate alphabet and can therefore complete an unassigned path. Injectivity gives unique decoding for occupied paths, while guaranteeing lexical completion would require a separate occupied-path constraint, such as trie-based prefix masking.

Reproducibility. Table 3 records the SHA-256 digests of the two base embedding-derived representation JSON resources. The permutation controls use additional derived JSON maps. Each run's result manifest records the SHA-256 digest of the particular map it uses, and the anonymous supplement contains the maps, manifests, and split metadata.

Table 3: SHA-256 digests of the base embedding-derived representation JSON resources. Derived permutation maps and their digests are retained in the anonymous supplement.

Corpus	Base representation JSON SHA-256
Tiny Shakespeare	`419e1d614d35dba1cfcd60c1a8045f6c5b40688b2aaf5bdb1633b0e39c3f6c80`
enwik8	`92948069cb03978f45e5edf42334546f53d0b0fcc116de80ef67865267d80fcf`

B Likelihood and Prefix-Aware Training

A lexical entry represented by a coordinate path is predicted through S successive coordinate decisions rather than one vocabulary-sized decision. This section first makes that likelihood factorization explicit and contrasts it with Atomic Lexical. It then gives the role-specific interfaces and prefix-aware training objective used by Full ILM.

B.1 Factorized and atomic lexical prediction

Both ILM and Atomic Lexical begin from the same lexical vocabulary $\mathcal{W}$, but they represent its entries differently. In the constructive representation, the map

$$\phi : \mathcal{W} \hookrightarrow \mathcal{A}_K^S$$

assigns each lexical entry w_n a path

$$\phi(w_n) = (c_0(w_n), \ldots, c_{S-1}(w_n)).$$

Because this path is emitted one coordinate at a time in the flattened stream, its probability factors into S successive next-coordinate predictions:

$$P_\theta(\phi(w_n) \mid x_{<Sn}) = \prod_{p=0}^{S-1} P_\theta(c_p(w_n) \mid x_{<Sn+p}) \,. \tag{17}$$

Each factor is normalized over the same coordinate alphabet $\mathcal{A}_K$. The first coordinate is therefore predicted from the preceding stream, while each later coordinate can additionally condition on the coordinates already observed within the same lexical path.

Atomic Lexical keeps the same vocabulary $\mathcal{W}$ but replaces the length-S representation with a one-event lexical representation. Let

$$\phi_{\text{lex}} : \mathcal{W} \xrightarrow{\cong} \mathcal{A}_{\text{lex}}, \qquad \mathcal{A}_{\text{lex}} = \{0, \ldots, |\mathcal{W}| - 1\}, \tag{18}$$

assign each lexical entry a unique value in a vocabulary-sized alphabet. Given a contextual state $h \in \mathbb{R}^C$, Atomic produces lexical logits

$$Z^{\text{lex}} = hW_{\text{lex}} + \beta_{\text{lex}}, \qquad W_{\text{lex}} \in \mathbb{R}^{C \times |\mathcal{W}|}, \qquad \beta_{\text{lex}} \in \mathbb{R}^{|\mathcal{W}|}, \tag{19}$$

and therefore assigns

$$P_\theta(w \mid h) = \operatorname{softmax}\big(Z^{\text{lex}}\big)_{\phi_{\text{lex}}(w)} \,. \tag{20}$$

Thus, the two models use the same lexical segmentation but differ in their prediction interface. Atomic Lexical maps an entry w to a single event $\phi_{\text{lex}}(w) \in \mathcal{A}_{\text{lex}}$ and makes one $|\mathcal{W}|$-way prediction. In contrast, ILM maps w to an ordered path $\phi(w) \in \mathcal{A}_K^S$ and makes S successive K-way predictions, so later coordinates can condition on earlier coordinates of the same path. Atomic Lexical therefore removes coordinate factorization while preserving the lexical vocabulary. Once lexical identity is instead represented by an ordered path, the positions $p \in \{0, \ldots, S-1\}$ provide an additional structure that the model can choose to represent explicitly.

B.2 ROLE-SPECIFIC INTERFACES

Flat ILM uses the same input table and output classifier at every position p within a lexical path. Full ILM instead exposes this path position explicitly through role-specific input and output interfaces.

For each batch element b, let

$$x_{b,0}, \ldots, x_{b,B}$$

be $B+1$ consecutive coordinate events. The first B events serve as context positions and predict the following B events. Let $r_b \in \{0, \ldots, S-1\}$ be the role of $x_{b,0}$. The event at local context position t has role

$$p_{b,t} = (r_b + t) \bmod S, \tag{21}$$

while its next target has role

$$p^+_{b,t} = (r_b + t + 1) \bmod S. \tag{22}$$

For each role $p \in \{0, \ldots, S-1\}$, Full uses an input table $U_p \in \mathbb{R}^{K \times C}$, an output matrix $W_p \in \mathbb{R}^{C \times K}$, and an output bias $\beta_p \in \mathbb{R}^K$. Let $H^{(0)}_{b,t}, H^{(L)}_{b,t} \in \mathbb{R}^C$ denote the input state and the state after L Transformer blocks, and let $\mathrm{PE}_t \in \mathbb{R}^C$ be the learned absolute positional embedding. The input state and next-coordinate logits are

$$H^{(0)}_{b,t} = U_{p_{b,t}}[x_{b,t}] + \mathrm{PE}_t, \tag{23}$$

$$Z_{b,t} = H^{(L)}_{b,t} W_{p^+_{b,t}} + \beta_{p^+_{b,t}}, \qquad 0 \le t < B. \tag{24}$$

Thus, the interface used to encode the current coordinate depends on its role within the lexical path, while the classifier used at that state depends on the role of the coordinate predicted next. In either case, $\mathrm{softmax}(Z_{b,t})$ remains a distribution over the same K coordinate values. Products of the corresponding next-coordinate probabilities recover the factorized likelihood in Equation 17.

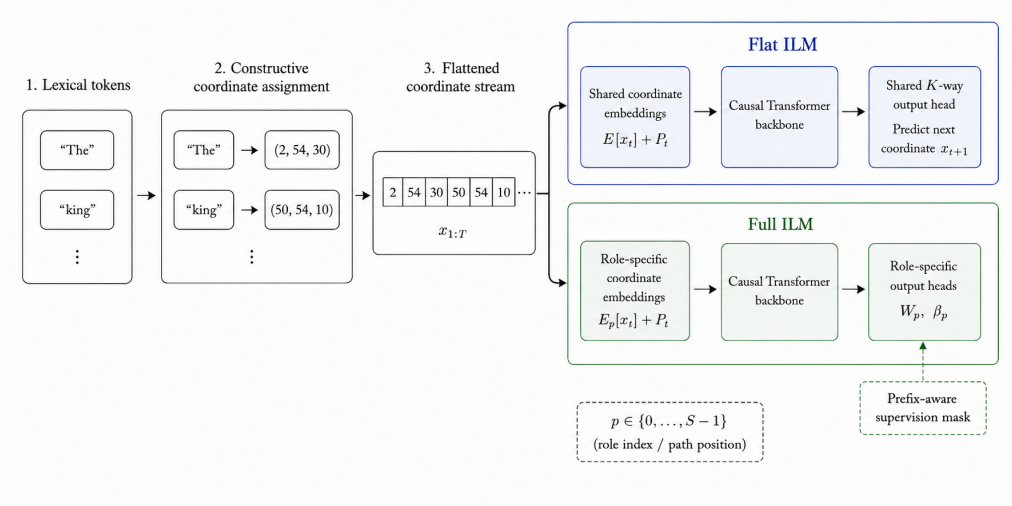


Figure 4: **Coordinate roles and model interfaces.** The displayed paths have $S = 3$. In the path $(2, 54, 30)$ for "The", the coordinate values 2, 54, and 30 have roles 0, 1, and 2, respectively. Blue boxes denote Flat ILM's shared interfaces, while green boxes denote Full ILM's role-specific interfaces. The dashed green box and arrow denote the prefix-aware training mask described in Equation 27.

B.3 PREFIX-AWARE TRAINING

Role-specific interfaces identify where each coordinate lies within a path, but sampled training windows need not begin at a path's first coordinate. When a window begins at role $r_b > 0$, its left edge contains only the suffix of a path whose earlier coordinates lie outside the window. Full therefore

excludes losses whose context states belong to this incomplete fragment and resumes supervision at the first path that begins inside the window.

The number of context positions in the incomplete left-edge fragment is

$$\delta_b = (S - r_b) \bmod S. \tag{25}$$

The prefix-aware mask over the B next-coordinate prediction positions is therefore

$$m_{b,t}^{\text{prefix}} = \begin{cases} 0, & 0 \le t < \delta_b, \\ 1, & \delta_b \le t < B. \end{cases} \tag{26}$$

Writing $y_{b,t} = x_{b,t+1}$ for the next-coordinate target, the Full training objective is

$$\mathcal{L}_{\text{prefix}} = -\frac{\sum_{b,t} m_{b,t}^{\text{prefix}} \log \text{softmax}(Z_{b,t})_{y_{b,t}}}{\sum_{b,t} m_{b,t}^{\text{prefix}}}. \tag{27}$$

For example, consider the five-event segment

$$(54, 30, 50, 54, 10),$$

which contains the visible suffix $(54, 30)$ of the path for "The" followed by the complete path $(50, 54, 10)$ for " king". The first four events are context positions that predict the following four events, so $B = 4$. Since the first context event has role $r_b = 1$ and $S = 3$, Equation 25 gives $\delta_b = 2$, and hence

$$m_b^{\text{prefix}} = (0, 0, 1, 1).$$

The first two losses are removed because their context states belong to the incomplete suffix of "The", whose first coordinate lies before the window. The event 50 begins a path that is fully visible from its start inside the window, so its context state is the first retained state and is used to predict the next coordinate, 54.

The prefix mask changes only which next-coordinate losses contribute during training. It does not alter causal attention, coordinate-by-coordinate generation, or test-time likelihood evaluation.

C Experimental and Reproducibility Details

Reproducing the reported BPB values requires fixing both the source bytes scored and the context available to each prediction. We specify these choices first, then record the model configurations, optimization protocols, and training diagnostics used in the comparisons.

C.1 Corpora and evaluation

For model fitting and evaluation, each corpus is divided into nonoverlapping training, validation, and test source regions, and each region is encoded independently. Tiny Shakespeare retains the order of the source text: the first 892,315 bytes form the training region, the next 2,038 bytes are left unused, the following 109,503 bytes form the validation region, another 2,060 bytes are left unused, and the final 109,478 bytes form the test region. The split procedure requested approximately 2,048 unused bytes at each boundary and then moved each held-out region to the next line start, which accounts for the two slightly different gap lengths. enwik8 uses its standard contiguous partition of 90 million training bytes, 5 million validation bytes, and 5 million test bytes. Encoding the regions independently ensures that every training or evaluation context contains source text from only one split.

BPB normalization. The representations emit different numbers of events for the same source text, so BPB normalizes their prediction losses by represented UTF-8 source bytes. Write u_j for the jth source unit before it is represented as model events. For a lexical model, u_j is one occurrence of a lexical entry. For a raw-symbol model, it is one character or byte. The representation converts these units into an event stream $(x_0, x_1, \ldots)$, where x_t is a coordinate, lexical category, character, or byte according to the model.

Suppose that u_j occupies b_j UTF-8 bytes and emits ν_j model events. Constructive lexical units have $\nu_j = S$, while Atomic Lexical and raw-symbol units have $\nu_j = 1$. Let $\mathcal{I}$ be the set of event positions selected for scoring by the evaluation procedure described below, and let ℓ_t be the teacher-forced negative log probability of event x_t for $t \in \mathcal{I}$, measured in nats. If x_t is emitted for source unit u_j, it receives source-byte weight $\omega_t = b_j/\nu_j$. The effective number of scored source bytes is $B_{\text{eff}} = \sum_{t\in\mathcal{I}} \omega_t$, and

$$\text{BPB} = \frac{\sum_{t\in\mathcal{I}} \ell_t}{\log 2 \sum_{t\in\mathcal{I}} \omega_t}. \tag{28}$$

The numerator sums prediction loss over scored events. The factor $\log 2$ converts this loss from nats to bits, while the remaining denominator converts it to a per-source-byte quantity. For a fully scored source unit, the weights of its ν_j events sum to its original byte length b_j.

Evaluation contexts. The two test regions use different teacher-forced evaluation procedures. Tiny Shakespeare uses rolling full-context evaluation, whereas enwik8 uses block-reset evaluation. This choice reflects the size of the held-out streams. Rolling evaluation is practical on Tiny Shakespeare and supplies each target with up to the latest B observed events. Applying the same overlapping window across enwik8's five-million-byte test region would require a separate overlapping context for almost every target. Block-reset evaluation instead obtains all B next-event losses in each nonoverlapping block from one forward pass, making exhaustive evaluation over the complete blocks tractable. Every model family within a corpus follows the same procedure. In both procedures, every input and target comes from the observed held-out stream, and sampled predictions are never fed back into the context.

Let $\mathcal{C}_t$ denote the observed context supplied when the model scores target event x_{t+1}. This target contributes

$$\ell_{t+1} = -\log P_\theta(x_{t+1} \mid \mathcal{C}_t) \tag{29}$$

to the numerator of Equation 28. Thus, the evaluation procedure affects BPB by determining the context inside each scored conditional probability.

For a context limit of B events, let k denote the block containing event x_t under block-reset evaluation, so $kB \le t < (k+1)B$. The two context rules are

$$\mathcal{C}_t^{\text{rolling}} = (x_{\max(0,t-B+1)}, \ldots, x_t), \tag{30}$$

$$\mathcal{C}_t^{\text{reset}} = (x_{kB}, \ldots, x_t). \tag{31}$$

The $+1$ in Equation 30 accounts for both included endpoints. Once $t \ge B-1$, enough events are available to fill the window, and the rolling context contains exactly B events.

Table 4 applies these rules to the coordinate models. Tiny Shakespeare uses $B = 60$, the event count of 20 paths of length $S = 3$, while enwik8 uses $B = 80$, the event count of 20 paths of length $S = 4$. Each target appears separately from the observed context on which its probability is conditioned in Equation 29.

Table 4: Teacher-forced contexts near the first window boundary. Both modes first build a causal history from x_0. Rolling evaluation then retains the latest events, whereas block-reset evaluation begins a new history at x_{80}.

	Tiny Shakespeare: rolling, $B = 60$		**enwik8**: block reset, $B = 80$	
Case	Observed context	Target	Observed context	Target
Initial step	x_0	x_1	x_0	x_1
First full window	$(x_0, \ldots, x_{59})$	x_{60}	$(x_0, \ldots, x_{79})$	x_{80}
After the boundary	$(x_1, \ldots, x_{60})$	x_{61}	x_{80}	x_{81}
One step later	$(x_2, \ldots, x_{61})$	x_{62}	(x_{80}, x_{81})	x_{82}

Both modes initially grow their causal history from the beginning of the test stream. Once the first window is full, rolling evaluation advances by one event and discards only the oldest event. Block-reset evaluation instead begins a new history at the next block boundary. The first predictions after this initial window illustrate the resulting difference:

$$\ell_{61}^{\text{rolling}} = -\log P_\theta(x_{61} \mid x_1, \ldots, x_{60}), \tag{32}$$

$$\ell_{81}^{\text{reset}} = -\log P_\theta(x_{81} \mid x_{80}). \tag{33}$$

If enwik8 used rolling evaluation with the same 80-event limit, its second expression would instead be

$$\ell_{81}^{\text{rolling}} = -\log P_\theta(x_{81} \mid x_1, \ldots, x_{80}). \tag{34}$$

The reset therefore changes the observed history inside ℓ_{81}, not the next-event likelihood being measured.

If an event stream has length N and $M = \lfloor (N-1)/B \rfloor$ complete reset blocks, the scored target sets are

$$\mathcal{I}_{\text{rolling}} = \{1, \ldots, N-1\}, \qquad \mathcal{I}_{\text{reset}} = \{1, \ldots, MB\}. \tag{35}$$

Rolling evaluation scores every target after x_0. Block-reset evaluation instead scores the targets in its complete blocks and omits the final incomplete suffix.

The context limits in Table 4 apply to the coordinate representations. The other model families retain the same corpus-specific procedure but use limits expressed in their own events. Character GPT uses 74 character events and Atomic Lexical uses 20 lexical events under the Tiny Shakespeare rolling rule. Byte GPT uses a 74-byte context under the enwik8 block-reset rule.

Two related mechanisms remain separate from these evaluation procedures. Full ILM's prefix-aware mask selects training losses and is inactive during evaluation. Free-running generation instead replaces observed held-out context events with events generated by the model.

An evaluation boundary can leave part of a coordinate path unscored. The set $\mathcal{I}$ and effective byte count B_{eff} therefore follow the exact event positions scored by the corresponding evaluation rule.

Table 5 illustrates the resulting weights. For a constructive entry, its byte length is divided across its S events. For an atomic entry or raw symbol, the single event receives the full byte length.

Table 5: Examples of the source-byte weights used in Equation 28. The lexical entry " king" includes its leading space.

Representation	Source unit u_j	b_j	ν_j	ω_t	$\sum_t \omega_t$
Tiny constructive ($S = 3$)	`The`	3	3	1	3
Tiny constructive ($S = 3$)	`␣king`	5	3	$5/3$	5
enwik8 constructive ($S = 4$)	`The`	3	4	$3/4$	3
enwik8 constructive ($S = 4$)	`␣king`	5	4	$5/4$	5
Atomic Lexical	`␣king`	5	1	5	5
Character or byte	one ASCII symbol	1	1	1	1

C.2 Models and optimization

Every model consumes a sequence of discrete tokens. The relevant distinction is between how a scanner or tokenizer segments the source text and how each resulting unit is encoded as model tokens. Table 6 separates these two choices. The subword row provides conceptual context and is not an evaluated baseline in this study.

Table 6: How scanner or tokenizer units are encoded as model tokens. Atomic Lexical and ILM use the same frozen lexical scanner, isolating the effect of factorizing lexical identity.

Model family	Scanner or tokenizer units	Model tokens
Character/Byte GPT	Individual characters or bytes	Same as units
Subword GPT	Learned variable-length pieces (e.g., BPE or SentencePiece)	Same as units
Atomic Lexical	Scanner-defined word-like spans, punctuation, whitespace, and line breaks	Same as units
Flat/Full ILM	Scanner-defined word-like spans, punctuation, whitespace, and line breaks	S sequential coordinates from an alphabet of size K

The experiments use two Transformer implementations. Atomic Lexical, Flat, Permuted Flat, and Full use the ILM implementation, whereas Character GPT and Byte GPT use nanoGPT (Karpathy, 2022). This distinction matters for the optimizer crossover because the crossover exchanges training protocols while leaving each model's architecture unchanged.

The reported ILM backbone uses pre-layer-normalized residual blocks. Each block has six causal attention heads with independent bias-free query, key, and value projections, followed by a biased output projection. Its MLP expands from C to $4C$, applies ReLU, and projects back to C. Dropout is applied to attention probabilities, attention outputs, and MLP outputs, and a final layer normalization precedes the language-model head. Coordinate and positional embedding parameters are distinct, and input embedding weights are not tied to the output-head weights. Every reported ILM variant shares this backbone. Full ILM differs only in its coordinate-role interfaces and training loss selection.

Character GPT and Byte GPT instead use nanoGPT's pre-layer-normalized blocks, GELU MLP, tied input and output token weights, and the bias-free linear and normalization layers selected by their configurations. Thus, the optimizer crossover changes the training protocol rather than replacing either model family's architecture.

Table 7 records the complete model matrix. The 6.5M tier contains all principal controls. Character GPT and Byte GPT appear there for the optimizer crossover, while Permuted Flat tests the lexical-to-code assignment across three assignment seeds and three training seeds. Atomic Lexical is evaluated only on Tiny Shakespeare because its vocabulary-sized interfaces can be parameter-matched there without leaving too little capacity in the Transformer backbone. At 15.5M, Atomic Lexical is repeated on Tiny Shakespeare, while Flat and Full are compared on both corpora. The 100M tier is restricted to the enwik8 Flat–Full comparison.

Table 7: Representation-specific configurations. Mean source bytes is the average amount of training-source text represented by one full input context. Permuted Flat includes unrestricted and exact-frequency assignments, which share Flat's architecture. All models use six blocks, six heads, batch size 32, dropout 0.5, and 6,000 updates.

Corpus	Family	Event alphabet	Context events	Mean bytes	Width	Parameters
Tiny Shakespeare	Character GPT (6.5M)	65 characters	74	74.0	300	6,525,600
Tiny Shakespeare	Atomic Lexical (6.5M)	15,030 entries	20	73.9	156	6,469,374
Tiny Shakespeare	Permuted Flat (6.5M)	64 coordinates, $S = 3$	60	73.9	300	6,555,064
Tiny Shakespeare	Flat ILM (6.5M)	64 coordinates, $S = 3$	60	73.9	300	6,555,064
Tiny Shakespeare	Full ILM (6.5M)	64 coordinates, $S = 3$	60	73.9	300	6,631,992
Tiny Shakespeare	Atomic Lexical (15.5M)	15,030 entries	20	73.9	300	15,537,630
Tiny Shakespeare	Flat ILM (15.5M)	64 coordinates, $S = 3$	60	73.9	462	15,483,532
Tiny Shakespeare	Full ILM (15.5M)	64 coordinates, $S = 3$	60	73.9	462	15,601,932
enwik8	Byte GPT (6.5M)	205 bytes	74	74.0	300	6,567,600
enwik8	Permuted Flat (6.5M)	64 coordinates, $S = 4$	80	72.2	300	6,561,064
enwik8	Flat ILM (6.5M)	64 coordinates, $S = 4$	80	72.2	300	6,561,064
enwik8	Full ILM (6.5M)	64 coordinates, $S = 4$	80	72.2	300	6,676,456
enwik8	Flat ILM (15.5M)	64 coordinates, $S = 4$	80	72.2	462	15,492,772
enwik8	Full ILM (15.5M)	64 coordinates, $S = 4$	80	72.2	462	15,670,372
enwik8	Flat ILM (100M)	64 coordinates, $S = 4$	80	72.2	1,176	99,891,856
enwik8	Full ILM (100M)	64 coordinates, $S = 4$	80	72.2	1,176	100,343,632

Table 8 records the complete protocols exchanged in the 6.5M optimizer crossover. Each protocol is applied to both the raw-symbol reference and the corresponding Flat model, so both representations are evaluated under each training procedure.

Table 8: Complete protocols used in the 6.5M optimizer crossover.

Setting	ILM protocol	nanoGPT protocol
Optimizer	AdamW	AdamW
Learning rate	10^{-3} constant	$10^{-3} \to 10^{-4}$ cosine
Warmup	none	100 updates
Weight decay	0.01, all tensors	0.1, matrices only
Adam betas	$(0.9, 0.999)$	$(0.9, 0.99)$
Gradient clipping	none	1.0

The 15.5M comparisons use only the ILM implementation and its constant-rate protocol. The optimizer crossover is confined to the 6.5M tier. All reported experiments used Python 3.9.6, PyTorch 2.6.0, NumPy 2.0.2, and scikit-learn 1.6.1 on Apple MPS. Run manifests retain resource hashes, tokenizer hashes, event counts, source-byte denominators, and per-run BPB results.

C.3 Results by training seed at 6.5M

Figure 5 shows the results behind the 6.5M means for each training seed in Table 2. The plot complements the exact table values by displaying variation across training seeds and the repeated Flat-to-Full ordering on both corpora.

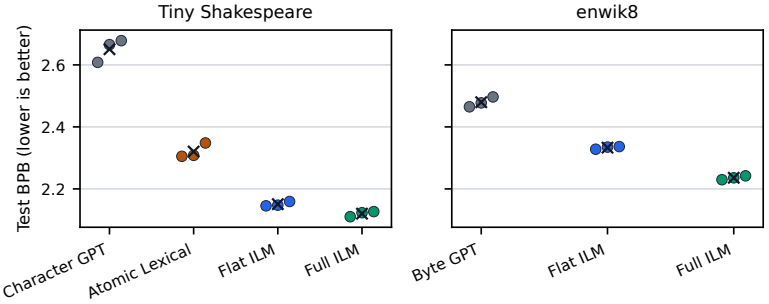


Figure 5: BPB by training seed at approximately 6.5M parameters. Circles show training-seed results, and black $\times$ markers show means. Corpus panels share a vertical range. Atomic Lexical appears only on Tiny Shakespeare because its vocabulary-sized interfaces would leave too little Transformer-backbone capacity at the matched enwik8 parameter budget.

C.4 100M extension on enwik8

The 100M extension on enwik8 tests whether the Flat–Full ordering persists at greater capacity. Both models use an 80-event context, which represents 20 complete four-coordinate lexical paths, and increase the Transformer width to 1,176. The resulting models contain 99,891,856 trainable parameters for Flat and 100,343,632 for Full. As in the compact experiments, every reported run is trained for 6,000 parameter updates. Their validation trajectories appear in Appendix C.7.

During a preliminary Flat-only run with training seed 13, we applied the constant-rate compact protocol in Table 8 at 100M parameters. Its native validation objective rose from approximately 3.24 at update 3,800 to 3.34 by update 6,000, indicating late-stage instability at that scale. We therefore trained the reported Flat and Full conditions with the shared warmup-and-cosine regime in Table 9. Only these subsequent runs enter the reported Flat–Full BPB comparison.

Table 9: Shared optimization regime for the 100M extension on enwik8.

Setting	100M large-scale cosine regime
Optimizer	AdamW
Learning rate	$3 \times 10^{-4} \rightarrow 3 \times 10^{-5}$ cosine
Warmup	200 updates
Weight decay	0.01, all tensors
Adam betas	$(0.9, 0.999)$
Gradient clipping	1.0
Training horizon	6,000 updates

Table 10 gives the complete held-out BPB summary over the three training seeds. It is the numerical source for the 100M points in Figure 2.

Table 10: Teacher-forced BPB for the 100M extension on enwik8. Values are mean $\pm$ sample standard deviation over training seeds 13, 29, and 47.

Family	Trainable parameters	BPB (mean $\pm$ s.d.)
Flat ILM	99,891,856	2.2100 ± 0.0021
Full ILM	100,343,632	2.1134 ± 0.0016

C.5 Contemporary context for absolute BPB

The optimizer crossover shows that implementation and training protocol can substantially change absolute BPB. Therefore, this section reports two external results that provide useful numerical context, rather than a comprehensive comparison or a basis for ranking systems trained under different conditions.

For Tiny Shakespeare, one recent point of reference comes from the 1GC-7RC benchmark, which uses a character-level language-modeling task and cites 2.120 BPB as a published nanoGPT reference (Kampa et al., 2026). Full ILM reaches 2.1201 ± 0.0086 BPB at the 6.5M tier. However, the 1GC-7RC task uses a different 90/10 train-validation split, and its authors explicitly present published reference values as context rather than as directly comparable experiments. Accordingly, the numerical proximity situates the absolute likelihood scale without establishing an ordering between the systems.

For enwik8, StateSMix provides complementary context from online lossless compression (Tacconelli, 2026). It reports 2.130 BPB on the complete 100 MB file using a Mamba predictor initialized from scratch and trained online, sparse n-gram context mixing, and arithmetic coding. Full ILM reaches 2.1134 ± 0.0016 held-out BPB at the 100M tier using an 80-event block-reset context after training on a separate region. Established offline, long-context models occupy a different numerical regime. For example, Transformer-XL reports 0.99 bits per character (BPC) under its enwik8 protocol (Dai et al., 2019). These values locate the present results among distinct training and evaluation settings rather than define a direct ranking.

C.6 Native validation-objective trajectories

Figure 6 records the native validation objectives for Flat and Full ILM over the common 6,000-update horizon. These curves diagnose optimization progress within each family. Flat ILM uses ordinary next-event cross-entropy, whereas Full ILM applies the prefix-aware loss mask of Equation 7. Because the two objectives select different prediction events, their absolute values are not a cross-family comparison metric.

Across the displayed compact runs, 89–94% of the net decrease from update 0 to update 6,000 occurs by update 1,000. The trajectories therefore characterize 6,000 updates as a common post-transient evaluation horizon.

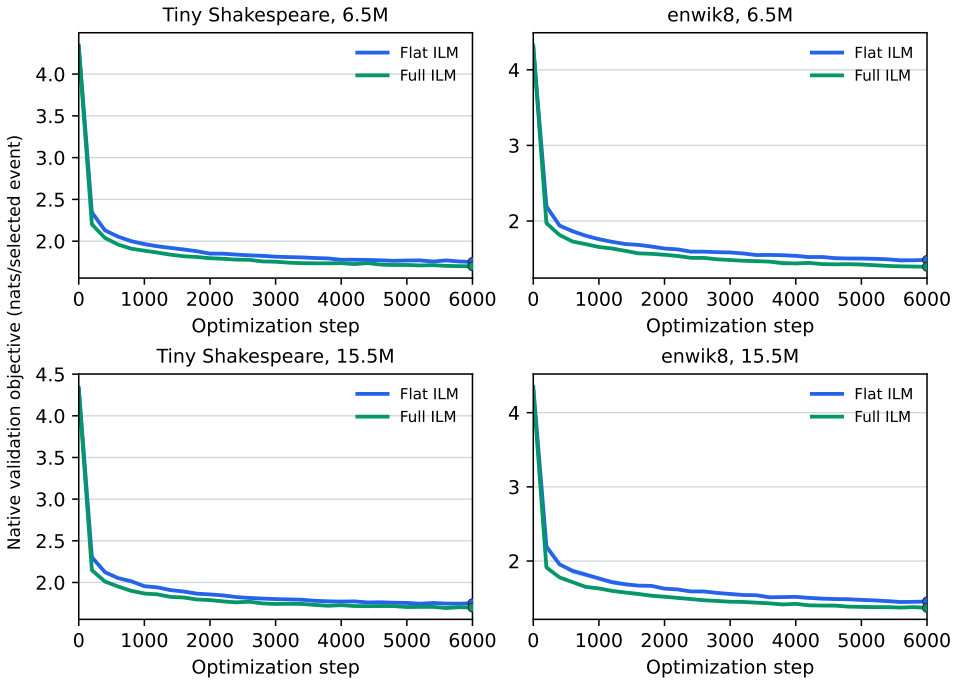


Figure 6: Native validation-objective trajectories over the common 6,000-update horizon for the 6.5M and 15.5M Flat and Full ILM runs. Curves show each model's native validation objective averaged over training seeds.

Figure 7 enlarges the final 2,000 updates so that the smaller late-stage changes remain visible.

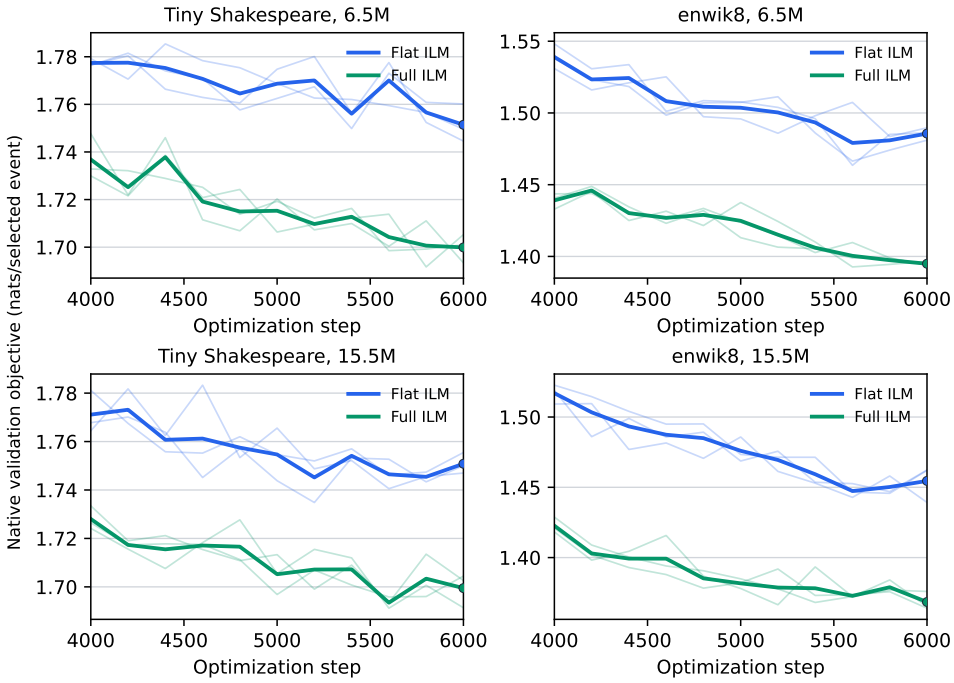


Figure 7: Final 2,000 updates of the native validation-objective trajectories in Figure 6. The enlarged view shows the smaller late-stage changes within each family. Faint curves show individual training seeds, while dark curves show their means.

C.7 100M NATIVE VALIDATION-OBJECTIVE TRAJECTORIES

Because late-stage instability in the preliminary Flat run motivated the separate 100M optimization regime, Figures 8 and 9 record the native validation objectives for the three-seed 100M extension on enwik8. Under the shared cosine regime, both families show a sharp initial reduction followed by a slower decrease over the common 6,000-update horizon. As in the compact runs, these objectives diagnose optimization within each family, while teacher-forced BPB provides the cross-family likelihood comparison.

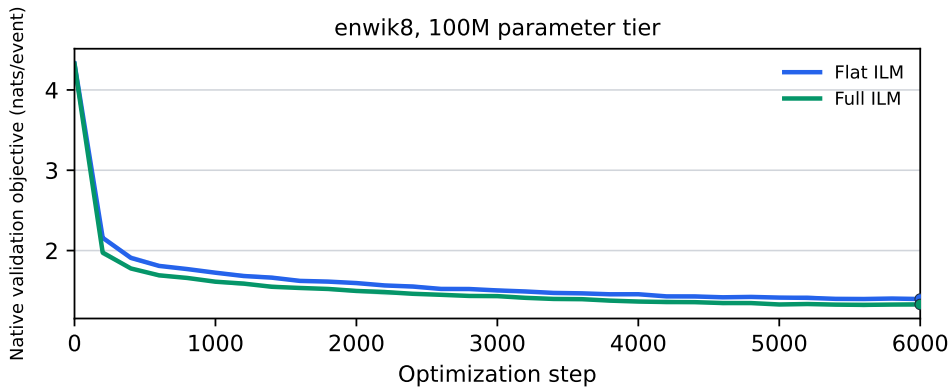


Figure 8: Native validation-objective trajectories for the 100M extension on enwik8. Faint curves show individual training seeds, and dark curves show their means.

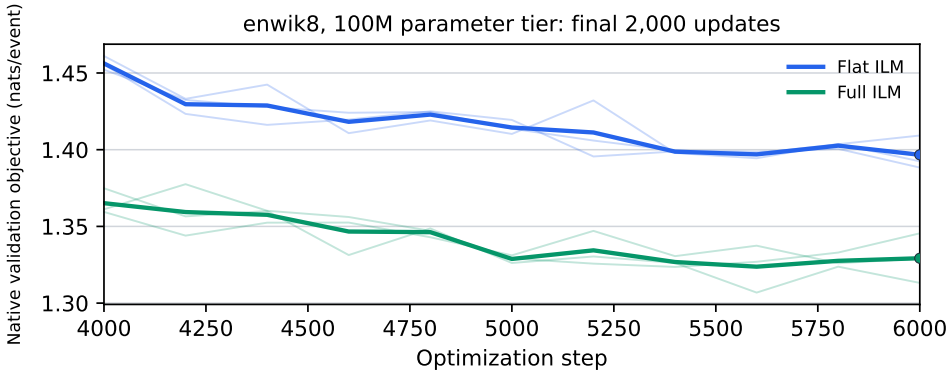


Figure 9: Final 2,000 updates of the 100M native validation-objective trajectories. The enlarged view records continued late optimization within each model family. Faint curves show individual training seeds, and dark curves show their means.

D Permutation Controls and Replicated Results

Section 6.1 summarizes how permutation changes predictive performance. To interpret that result, this appendix distinguishes variation in the sampled assignment from variation in model training and quantifies what each reassignment changes. Equation 8 defines the reassigned map ϕ_{π_r}, where r indexes the assignment seed. We use assignment seeds 314159, 271828, and 161803, and evaluate every resulting map with training seeds 13, 29, and 47. The two controls use the same three assignment seeds but sample different maps under their respective constraints.

Here each $w \in \mathcal{W}$ is one distinct entry in the frozen vocabulary, such as "The" or " king." An occurrence is one appearance of that entry in a data split. We can therefore measure both how much of the vocabulary map changes and how much observed text is carried by the changed entries.

D.1 Measuring a reassignment

We use complementary diagnostics to describe how strongly a permutation changes the lexical-to-code map. In the formulas below, ϕ_π denotes any one sampled reassignment. For a data split $D \in \{\text{train}, \text{test}\}$, let $f_D(w)$ be the number of occurrences of lexical entry w in that split, and let $F_D = \sum_{w \in \mathcal{W}} f_D(w)$ be the total number of lexical-entry occurrences in the split.

First, the moved-entry fraction counts every distinct vocabulary entry once. We denote it by τ_π and define it as

$$\tau_\pi = \frac{1}{|\mathcal{W}|} \sum_{w \in \mathcal{W}} \mathbf{1}\{\phi_\pi(w) \neq \phi(w)\}\,. \tag{36}$$

Thus, τ_π is the fraction of entries that receive a different path, regardless of how often those entries occur. The moved occurrence mass instead asks what fraction of split D is carried by the reassigned entries:

$$\rho_D = \frac{1}{F_D} \sum_{w \in \mathcal{W}} f_D(w)\, \mathbf{1}\{\phi_\pi(w) \neq \phi(w)\}\,. \tag{37}$$

Thus, $\rho_D = 0$ means that no lexical-entry occurrence receives a different path, whereas $\rho_D = 1$ means that every occurrence does.

Second, we measure how reassignment changes the frequency of coordinate values at each path position. Using $c_p(w)$ for the value at role p in the original path from Equation 1, the occurrence-weighted share of coordinate value k is

$$q_{D,p}(k) = \frac{1}{F_D} \sum_{w \in \mathcal{W}} f_D(w)\, \mathbf{1}\{c_p(w) = k\}\,. \tag{38}$$

This quantity is the fraction of lexical-entry occurrences in split D whose path has value k at role p. It is called a marginal distribution because it describes one role without retaining the other coordinates

in the path. If $c_p^\pi(w)$ denotes role p of the reassigned path $\phi_\pi(w)$, then $q_{D,p}^\pi(k)$ is defined by replacing $c_p(w)$ with $c_p^\pi(w)$ in Equation 38.

Third, total-variation distance summarizes the difference between the original and reassigned coordinate-value distributions. We compute it for every role and report the largest role-wise change:

$$\mathrm{TV}_D = \max_{0 \le p < S} \frac{1}{2} \sum_{k \in \mathcal{A}_K} \left| q_{D,p}(k) - q_{D,p}^\pi(k) \right|. \tag{39}$$

A value of zero means that reassignment leaves the frequency distribution at every role unchanged. Exact-frequency reassignment has $\mathrm{TV}_{\mathrm{train}} = 0$ by construction. Its test value can be nonzero because entries that occur equally often in training need not occur equally often in the test split.

Total variation also has a direct probability-mass interpretation. At a fixed role, it is the smallest fraction of occurrence probability that must be moved between coordinate values to transform one marginal distribution into the other. Since Equation 39 takes the maximum over roles, TV_D reports this fraction at the role changed most by the reassignment. The three diagnostics therefore answer different questions: τ_π measures how broadly the map changes across vocabulary entries, ρ_D measures how much of split D is carried by those entries, and TV_D measures how strongly their reassignment changes the occurrence-weighted coordinate frequencies.

As a concrete $S = 3$ example, suppose a toy training split contains four occurrences of "The", four of " king", and twelve of "." Their paths are $(2, 54, 30)$, $(50, 54, 10)$, and $(7, 7, 7)$, respectively. An exact-frequency reassignment may exchange the paths of "The" and " king" because both entries occur four times. If "." is the only entry occurring twelve times, its path cannot move under this rule. Two of the three distinct entries change paths, so Equation 36 gives $\tau_\pi = 2/3$. Those two entries supply eight of the twenty lexical occurrences. Consequently, Equation 37 gives $\rho_{\mathrm{train}} = 8/20 = 0.4$.

Before this exchange, the first path position contains value 2 four times, value 50 four times, and value 7 twelve times. Exchanging two paths carried by equally frequent entries leaves those counts unchanged. The same reasoning applies at the other two positions, so Equation 39 gives $\mathrm{TV}_{\mathrm{train}} = 0$. By contrast, an unrestricted exchange of the paths for "The" and "." changes the first-position counts to twelve for value 2, four for value 50, and four for value 7. The total-variation distance at that role is then $\frac{1}{2}(|4 - 12| + |12 - 4|)/20 = 0.4$.

D.2 Unrestricted Permuted Flat

For Unrestricted Permuted Flat, we sample a bijection of the occupied path set as defined in Equation 8. Every sampled map therefore satisfies

$$\mathrm{im}(\phi_{\pi_r}) = \mathrm{im}(\phi). \tag{40}$$

The occupied lexical paths and the Flat model remain fixed, while lexical entries are reassigned among those paths. Since the same paths remain occupied, the number of vocabulary entries using each coordinate value at a given role is unchanged. However, frequent and infrequent entries can exchange paths, so the occurrence-weighted distributions in Equation 38 can change.

Table 11 summarizes the assignment-level means and BPB penalties. Table 12 then reports the complete unrestricted matrix over assignment seeds and training seeds.

Table 11: Assignment-level summary for Unrestricted Permuted Flat at the 6.5M tier. Each assignment row averages training seeds 13, 29, and 47. Each cell reports Permuted Flat BPB followed by $\Delta = \mathrm{BPB}_{\mathrm{Permuted}} - \mathrm{BPB}_{\mathrm{Flat}}$. The final row averages the three assignment-level means. The Flat reference is the mean over the same three training seeds.

Assignment seed	Tiny Shakespeare Permuted BPB / Δ	enwik8 Permuted BPB / Δ
314159	2.2966 / +0.1458	2.5041 / +0.1710
271828	2.2940 / +0.1432	2.5210 / +0.1879
161803	2.2945 / +0.1437	2.5194 / +0.1863
Mean across assignments	2.2950 / +0.1442	2.5148 / +0.1817

Table 12: BPB by assignment seed and training seed for Unrestricted Permuted Flat at the 6.5M tier. The first column identifies the original Flat ILM condition or assignment seed r, and the columns under each corpus give training seed s. Each permuted run is paired with the Flat run sharing the same s. Smaller BPB indicates better held-out prediction.

	Tiny Shakespeare			enwik8		
Assignment	$s = 13$	$s = 29$	$s = 47$	$s = 13$	$s = 29$	$s = 47$
Flat ILM	2.159404	2.147534	2.145441	2.336511	2.334855	2.328025
314159	2.289941	2.309861	2.289881	2.493390	2.505747	2.513156
271828	2.294844	2.309640	2.277503	2.521699	2.519413	2.521927
161803	2.291491	2.300090	2.291818	2.520168	2.506005	2.531988

D.3 Exact-frequency Permuted Flat

Unrestricted reassignment can change the training coordinate frequencies just defined. Exact-frequency Permuted Flat removes that change. For each count $m \geq 0$, let

$$\mathcal{W}_m = \{w \in \mathcal{W} : f_{\text{train}}(w) = m\} \tag{41}$$

be the set of entries that occur exactly m times in training. We permute entries only within each $\mathcal{W}_m$, so every entry receives a path previously carried by an equally frequent entry. If $\mathcal{W}_m$ contains only one entry, that entry is invariant because it has no equally frequent partner. The class $\mathcal{W}_0$ contains entries absent from the training split. Such entries can move and contribute to τ_π, but their zero frequency means that they contribute nothing to ρ_{train}. Equivalently, each occupied path receives an entry with the same training count as its original entry, so every complete-path count is unchanged. Because the count of a prefix is the sum of the counts of the occupied paths that extend it, all within-path prefix counts and conditional next-coordinate frequencies are also preserved. Moreover, $q^{\pi}_{\text{train},p}(k) = q_{\text{train},p}(k)$ for every role p and value k, and $\text{TV}_{\text{train}} = 0$.

Table 13 gives the complete matrix over assignment seeds and training seeds for this stricter control.

Table 13: BPB by assignment seed and training seed for Exact-frequency Permuted Flat at the 6.5M tier. The first column identifies the original Flat ILM condition or assignment seed r, and the columns under each corpus give training seed s. Each permuted run is paired with the Flat run sharing the same s. Smaller BPB indicates better held-out prediction.

	Tiny Shakespeare			enwik8		
Assignment	$s = 13$	$s = 29$	$s = 47$	$s = 13$	$s = 29$	$s = 47$
Flat ILM	2.159404	2.147534	2.145441	2.336511	2.334855	2.328025
314159	2.252556	2.250825	2.255763	2.421674	2.432248	2.429661
271828	2.257908	2.243563	2.254899	2.419240	2.433407	2.431671
161803	2.269857	2.261934	2.266745	2.430255	2.430292	2.431270

D.4 Comparison of the two permutation controls

Figure 10 places the two controls on a common scale. In the figure, *Original assignment* denotes Flat ILM with the embedding-derived map ϕ. The *Exact-frequency* and *Unrestricted permutation* conditions denote the two Permuted Flat variants defined above.

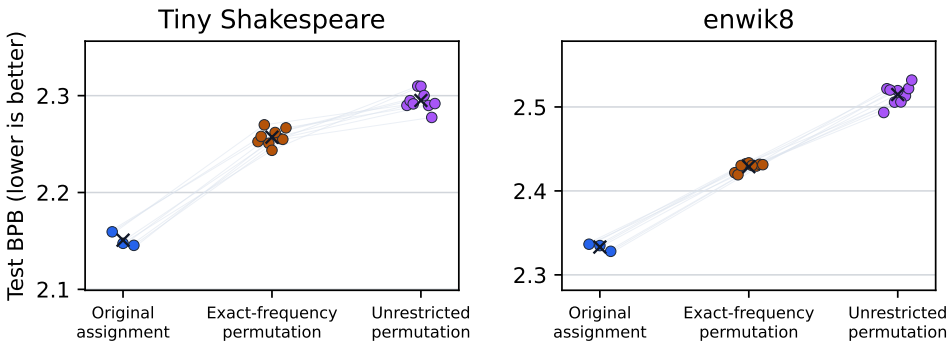


Figure 10: Frequency-control progression at the 6.5M tier. Lines connect conditions with the same numerical assignment seed and training seed. Circular markers show individual runs, and black $\times$ markers show condition means.

Both reassignment rules worsen BPB relative to the original assignment. Table 14 summarizes the assignment-level means and penalties.

Table 14: BPB mean and sample standard deviation across three assignment-level means, each averaged over training seeds 13, 29, and 47. The increase is relative to the Flat ILM mean over the same training seeds. All nine paired differences are positive for each control and corpus. The two standard deviations coincide because the Flat reference is fixed across assignments.

Corpus	Control	BPB (mean $\pm$ s.d.)	BPB increase over Flat (mean $\pm$ s.d.)
Tiny Shakespeare	Exact-frequency	2.2571 ± 0.0079	$+0.1063 \pm 0.0079$
	Unrestricted	2.2950 ± 0.0014	$+0.1442 \pm 0.0014$
enwik8	Exact-frequency	2.4289 ± 0.0015	$+0.0957 \pm 0.0015$
	Unrestricted	2.5148 ± 0.0093	$+0.1817 \pm 0.0093$

The controls differ in both moved occurrence mass and the resulting changes to coordinate-frequency distributions. Table 15 reports the diagnostics defined above.

Table 15: Displacement diagnostics averaged over three reassignment maps. The τ_π column gives the fraction of distinct vocabulary entries whose paths change in Equation 36. The ρ columns give moved occurrence mass from Equation 37, and TV gives the maximum role-wise distance from Equation 39.

Corpus	Control	τ_π	ρ_{train}	ρ_{test}	TV_{train}	TV_{test}
Tiny Shakespeare	Exact-frequency	0.981	0.313	0.314	0.000	0.049
	Unrestricted	1.000	1.000	1.000	0.626	0.638
enwik8	Exact-frequency	0.996	0.235	0.228	0.000	0.021
	Unrestricted	1.000	1.000	1.000	0.729	0.741

Table 15 separates the effects measured by these diagnostics. Unrestricted reassignment moves essentially the entire vocabulary and occurrence mass. At its most affected role, the mean training TV is 0.626 on Tiny Shakespeare and 0.729 on enwik8. Thus, matching the reassigned coordinate marginals to the original ones would require moving at least 62.6% and 72.9% of the respective role-wise probability mass.

Exact-frequency reassignment changes the map almost as broadly, moving approximately 98% of Tiny Shakespeare entries and more than 99% of enwik8 entries. However, many moved entries are rare or absent from training, while some high-frequency entries remain invariant because they are the sole members of their frequency classes. The moved occurrence mass is consequently lower, at

0.313 and 0.235. The corresponding test masses, 0.314 and 0.228, are similar, so the moved entries cover comparable shares of the training and held-out streams. More importantly, the exact-frequency constraint makes the training TV exactly zero, yet the control still worsens BPB on both corpora. Changes in training-weighted coordinate marginals are therefore not required for the permutation penalty.

The exact-frequency test TV is small but nonzero, at 0.049 on Tiny Shakespeare and 0.021 on enwik8. This difference remains because entries with equal training counts can have different test counts. The control therefore fixes the coordinate marginals that shape training while separately reporting the modest held-out marginal changes.