TOKEN MARKOV GRAPHS FOR DATA-EFFICIENT LANGUAGE MODEL PRETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

The cost of pretraining a language model is set by the tokens, and the wall-clock time, needed to reach a target loss, and high-quality text is becoming the binding resource. We introduce the *Token Markov Graph* (TMG), a directed graph of token-transition counts computed once from the pretraining corpus, and inject it into pretraining as an annealed graph-distillation loss or as a lightweight architectural memory module. TMG reduces the tokens needed to reach the baseline's final validation loss at and below the Chinchilla-optimal budget. At the Chinchilla-optimal budget, the architectural route (engram v1) needs 8.1% fewer tokens and 5.1% less wall-clock time, with a gain in all 9 seeds ($t(8) = 7.61$, 4.56, and 3.05 for tokens, wall-clock time, and final loss gap). At a fixed 100M-token budget, curvature-weighted distillation saves 14.2–20.0% of tokens for models from 172M to 3B parameters (minimum $t = 4.83$ on the loss gap), with token savings, and wall-clock savings of up to 20.9%, growing with model size. The gains hold under a tuned baseline learning rate, on two independently built graphs, and on code-only training (6 seeds, $t(5) = 3.96$), and carry over to next-word cloze completion. Because the graph is built once and reused across models and runs, corpus transition statistics offer an inexpensive, broadly applicable accelerant for pretraining.

1 INTRODUCTION

Training compute for language models has grown by orders of magnitude (Kaplan et al., 2020; Hoffmann et al., 2022; Brown et al., 2020), with matching financial and environmental cost (Strubell et al., 2019; Patterson et al., 2021), and high-quality text is approaching the size of current training sets (Villalobos et al., 2024; Muennighoff et al., 2023). Both constraints are governed by the tokens, and wall-clock time, a model needs to reach a given loss. Reducing it lowers the cost of every pretraining run: standard compute-optimal runs (Hoffmann et al., 2022), data-constrained runs (Muennighoff et al., 2023), the many shorter runs of data-mixture, hyperparameter, and architecture search (Xie et al., 2023), and continued pretraining.

Existing efficiency methods address this quantity only indirectly. Scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022) prescribe how to split a budget between parameters and tokens, not how to converge faster within it; data-mixture optimization (Xie et al., 2023) changes which examples are seen; knowledge distillation (Hinton et al., 2015) requires a trained teacher; and retrieval augmentation (Khandelwal et al., 2020; Borgeaud et al., 2022) targets inference-time perplexity. None uses information fixed by the corpus before training, yet much of what a model learns early is such information (which tokens follow which, and how often) and can be obtained by counting. We ask whether supplying it as a graph computed once from the corpus reduces the tokens needed to reach a target loss.

This design has three advantages. Transition statistics cost one counting pass yet occupy early training, so supplying them trades a one-time CPU cost for GPU steps in every later run. A graph exposes what single probabilities do not (bottleneck tokens, multi-step continuations, token communities), indicating where corpus statistics are most informative. And the signal is fixed before training and read with one lookup per step, needing no teacher, datastore, extra data, or data-pipeline change. The benefit compounds: every run on the corpus reuses the graph, and the savings we observe grow with model size up to 3B parameters.

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 introduce the *Token Markov Graph* (TMG), a directed graph of corpus bigram counts over the vocabulary: a compact summary whose size is bounded by the vocabulary and whose statistics saturate as the corpus grows, so it is built once and reused. From it we derive an Ollivier–Ricci-type curvature, multi-hop reachability, community structure, and higher-order routes at high-entropy tokens (Figure 1), and we evaluate twelve mechanisms that inject them into training, as distillation losses, initializations, or an architectural memory module (Figure 2), on two graphs, four model sizes from 172M to 3B parameters, and budgets up to the Chinchilla-optimal one.

Contributions. (1) TMG and graph-derived training signals with a one-time, amortized construction cost (Sections 3–4). (2) In Chinchilla-optimal training, engram v1 reaches the baseline's final loss with 8.1% fewer tokens and 5.1% less wall-clock time, gaining in all 9 seeds (Section 5.3). (3) At a 100M-token budget, curvature-weighted distillation saves 14.2–20.0% of tokens, with savings (up to 20.9% wall-clock at 3B) growing with model size (Sections 5.2, 5.4). (4) The gains hold under a tuned learning rate, across graphs, and on code, and appear in next-word cloze completion, detected by a sample-size-weighted aggregate (Sections 5.5–5.7).

2 Related Work

Data-efficient pretraining. Scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022) prescribe a compute-optimal allocation ($\sim$20 tokens per non-embedding parameter), and Muennighoff et al. (2023) study data-constrained scaling. Pretraining cost (Strubell et al., 2019; Patterson et al., 2021) has motivated methods that act on the data, through domain-mixture optimization (Xie et al., 2023), deduplication (Lee et al., 2022), pruning (Sorscher et al., 2022), or curricula (Bengio et al., 2009), or on the objective, through replaced-token detection (Clark et al., 2020) or multi-token prediction (Gloeckle et al., 2024). TMG keeps the data, their order, and the next-token objective, adding a signal from the same corpus, and is complementary to data-side methods.

Distillation and target smoothing. Knowledge distillation (Hinton et al., 2015), used to pretrain small open models (Gemma Team et al., 2024), requires training and running a larger teacher; TMG's loss-based route derives soft targets from corpus counts in a single pass. Label smoothing (Szegedy et al., 2016; Müller et al., 2019) and its unigram variants soften targets without context; TMG's targets are conditioned on the preceding token and weighted by graph structure, and only the latter improves on the baseline (Section 5.2).

n-gram statistics. Smoothed n-gram models (Kneser & Ney, 1995) preceded neural language models (Bengio et al., 2003). N-Grammer (Roy et al., 2022) adds latent n-gram embeddings to Transformers, Infini-gram (Liu et al., 2024) builds unbounded-n models over trillion-token corpora, and Li et al. (2022) fit a neural model to an n-gram model's residual, keeping the n-gram model in the predictor. Networks learn low-order statistics first (Belrose et al., 2024), Transformer predictions are often approximated by n-gram rules (Nguyen, 2024), and induction heads emerge later (Olsson et al., 2022), so the statistics learned first are available before training. TMG's loss route leaves no trace in the deployed model and supplies graph properties such as curvature, not only n-gram probabilities (Section 3.2).

Memory, knowledge graphs, and graph methods. kNN-LM (Khandelwal et al., 2020), RETRO (Borgeaud et al., 2022), and Memorizing Transformers (Wu et al., 2022) retrieve from an external store at inference. Product-key memory (Lample et al., 2019) and Engram (Cheng et al., 2026) add learned lookup tables, the latter as a sparsity axis complementary to Mixture-of-Experts (Shazeer et al., 2017); our architectural route adapts Engram to dense pretraining, addressing its memory with the corpus graph. Knowledge-enhanced pretraining (Zhang et al., 2019; Yasunaga et al., 2022) injects curated external knowledge graphs; TMG is built from the training corpus. Co-occurrence statistics underlie word embeddings (Mikolov et al., 2013; Pennington et al., 2014) and graph neural networks for text (Kipf & Welling, 2017; Yao et al., 2019); we use curvature (Ollivier, 2009; Topping et al., 2022), personalized-PageRank propagation (Gasteiger et al., 2019), and community detection (Blondel et al., 2008) to extract pretraining signals.

Novelty. Prior approaches modify the data, require a teacher, external knowledge, or an inference-time datastore, or keep an n-gram model in the predictor. To our knowledge, TMG is the first method that (i) derives topological training signals (curvature, multi-hop reachability, community structure) from the corpus's token-transition graph, (ii) injects them into pretraining at a one-time

cost, through a training-only loss or a lightweight module, and (iii) shows iso-loss savings in tokens and wall-clock time across model sizes, domains, and budgets up to the Chinchilla-optimal one.

3 TOKEN MARKOV GRAPH

3.1 DEFINITION

The Token Markov Graph is a directed graph over the vocabulary whose edge weights are corpus bigram counts, so that each token's outgoing edges approximate the next-token distribution a language model must otherwise learn. Let V be the vocabulary and $\mathcal{D}$ the corpus, viewed as one token stream. TMG is the weighted graph $G = (V, E, w)$, $E \subseteq V \times V$, with

$$w(u, v) \;=\; \big|\{\, i : x_i = u,\; x_{i+1} = v \,\}\big|, \qquad (x_i)_{i=1}^{|\mathcal{D}|} = \mathcal{D}, \tag{1}$$

i.e. $w(u, v)$ counts how often v immediately follows u; normalizing $w(u, \cdot)$ gives the maximum-likelihood estimate of $P(x_{i+1} = v \mid x_i = u)$, which a model's next-token predictor must approximate locally. We hypothesize that supplying it directly reduces the gradient steps needed to learn it.

For each u we keep its top-K successors $N_K(u) = \operatorname{argtop}_K^v w(u, v)$ in a fixed-width tensor, so that a training step's lookup is a single batched GPU gather by token id, and derive

$$p_{\text{raw}}(v \mid u) = \frac{w(u, v)}{\sum_{v' \in N_K(u)} w(u, v')}, \tag{2}$$

$$p_{\text{inv}}(v \mid u) \propto \frac{p_{\text{raw}}(v \mid u)}{\max(f(v), \tau)}, \tag{3}$$

$$c(u) = \exp\big(-H(u)\big) \cdot \mathbb{1}[\deg_K(u) > 0], \tag{4}$$

where $f(v)$ is the corpus count of v, $\tau = 1000$ is a count floor, and $\deg_K(u) = |N_K(u)| \le K$. $H(u)$ is the entropy of u's *full* successor distribution before truncation (the full out-degree averages ≈5570 on the 350B graph and is used for hub selection, Appendix A.5), which measures how weakly u constrains the next token; hence the mean entropies in Table 4 (3.87–4.37 nats) exceed $\ln 6 \approx 1.79$. p_{raw} is the empirical successor distribution; p_{inv} downweights frequent successors, counteracting G's bias toward punctuation and function words; and c is a *graph confidence*, near 1 for concentrated successor distributions and near 0 for diffuse or absent ones. Section 4 uses p_{inv} as the distillation target (p_{raw} only in ablations) and c as a per-position weight.

In the 350B graph (Figure 1), "New" is followed mostly by "York" ($p_{\text{raw}} = 0.64$), for which p_{raw} ranks a comma first; p_{inv} moves the mass to "City" and "Times" ("Times": $0.18 \to 0.71$), and paths such as "New" → "York" → "City" are what multi-hop propagation aggregates. Construction is one streaming counting pass (27.9 minutes per billion tokens on 4 cores, parallelizing linearly), performed once per corpus.

Compression and saturation. G's size is bounded independently of $|\mathcal{D}|$: at most $|V|^2$ edges, and the table read during training has exactly $|V| \cdot K$ entries (262,144×6, 25MB), more than four orders of magnitude smaller than the 350B-token stream it summarizes. Each row $p_{\text{raw}}(\cdot \mid u)$ converges as the count of u grows, so G saturates: additional corpus changes it less and less, frequent tokens first. Our two graphs show this: a 5.83× larger corpus yields only 3.3× more unique edges, and the agreement of their stored successor lists rises steadily with token frequency (Appendix A.4). A graph built once from a moderate corpus thus captures most of the signal TMG uses.

3.2 BEYOND A FIRST-ORDER MARKOV CHAIN

G's edge weights define a first-order Markov model, which cannot represent dependencies beyond the preceding token. This holds for p_{raw} and p_{inv}, but not for the derived quantities of Section 3.3, which are properties of the graph's topology. Multi-hop propagation (Equation 7) aggregates transition mass over paths of length up to H, approximating a higher-order chain; curvature (Equation 6) compares the successor distributions of adjacent tokens to identify structural bottlenecks (Topping et al., 2022), which no single transition probability expresses; community structure aggregates edges

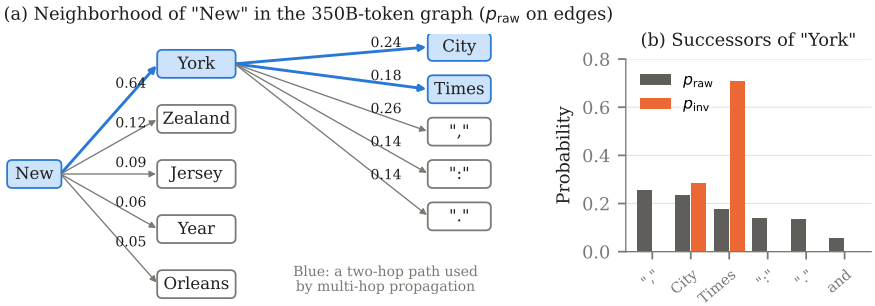


Figure 1: A real neighborhood of the 350B-token TMG. (a) Top-5 successors of "New" and five of the top-6 successors of "York" (p_{raw} on edges); the blue two-hop path is aggregated by multi-hop propagation. (b) p_{raw} versus p_{inv} for all six stored successors of "York": reweighting moves mass from punctuation to "City" and "Times".

by Louvain cluster (Blondel et al., 2008); and hub routing conditions on a four-token window at high-entropy positions. Consistently, curvature, the most topological quantity, yields the largest effect (Section 5.2).

3.3 Derived graph properties

Curvature. Tokens with equal confidence can differ in whether they lie at a structural bottleneck. We measure this with a heuristic based on Ollivier–Ricci curvature (Ollivier, 2009): treating the top-K successor distributions of i and j as distributions under a 0/1 ground metric, for which the Wasserstein-1 distance reduces to total variation, we define for an edge $i \to j$

$$\kappa(i,j) = 1 - 2\,\mathrm{TV}\big(p_{\text{raw}}(\cdot \mid i), p_{\text{raw}}(\cdot \mid j)\big), \qquad \mathrm{TV}(\mu,\nu) = 1 - \sum_{v} \min\big(\mu(v),\nu(v)\big), \tag{5}$$

$$b(u) = \max\Big(0,\, -\frac{1}{|N_K(u)|} \sum_{j \in N_K(u)} \kappa(u,j)\Big). \tag{6}$$

The factor of two (the standard discrete form $1 - \mathrm{TV}$ lies in $[0,1]$) gives $\kappa \in [-1,1]$; Equation 5 is thus an inexpensive proxy that avoids optimal transport, not the general Ollivier–Ricci curvature. Negative values mark edges whose endpoints have sharply different successor distributions, the bottlenecks studied by Topping et al. (2022); $b(u)$, the negative part of u's mean outgoing curvature, is positive only at bottleneck tokens.

Multi-hop propagation. A single hop misses tokens reachable through an intermediate step (*e.g.* a transitive subword continuation). With P the row-stochastic, N_K-restricted matrix with rows $p_{\text{raw}}(\cdot \mid u)$, we propagate for H hops with decay $\beta \in (0,1)$, in the spirit of personalized-PageRank propagation (Gasteiger et al., 2019):

$$p_{\text{multihop}}(\cdot \mid u) \;\propto\; \mathrm{top}_K\Big(\sum_{h=1}^{H} \beta^{h-1} P^h(u,\cdot)\Big), \tag{7}$$

re-truncating each intermediate power to its top-K entries to keep every hop sparse, then renormalizing the final sum.

Community structure and hub routing. We also compute a Louvain community assignment π with a community-level transition matrix Q, and a higher-order route table ρ at high-entropy hub tokens, used by mechanisms evaluated on the 60B graph (Appendix A.5).

Frequency weight. As a static, topology-free alternative to $c(u)$, we define

$$w_{\text{freq}}(u) \;=\; \mathrm{clip}\Big(\big(\tilde{f}/f(u)\big)^{\gamma},\, w_{\min}, w_{\max}\Big), \tag{8}$$

with $\tilde{f}$ the median vocabulary count, γ a fixed exponent, and $w_{\min}, w_{\max}$ bounds; it uses only G's node frequencies.

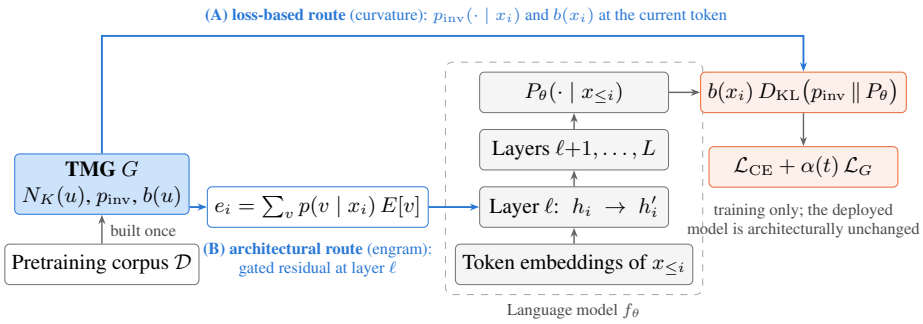


Figure 2: The two routes for injecting TMG, read at the current token x_i. (A) Loss-based (curvature): $p_{\text{inv}}(\cdot \mid x_i)$ is distilled into the prediction, weighted by $b(x_i)$ and annealed by $\alpha(t)$; training only. (B) Architectural (engram): the successor embedding e_i enters layer ℓ through a gated residual.

4 TMG-Assisted LLM Pretraining

4.1 General formulation

Let f_θ be the base model and $\mathcal{L}_{\text{CE}}(\theta; x)$ the next-token cross-entropy on a sequence x. We inject G through one of two routes (Figure 2): a loss-based route, which adds an auxiliary term and leaves the deployed model unchanged, and an architectural route, which feeds the graph into an intermediate layer. The loss-based route optimizes

$$\mathcal{L}_{\text{total}}(\theta; x, t) \;=\; \mathcal{L}_{\text{CE}}(\theta; x) \;+\; \alpha(t) \cdot \mathcal{L}_G(\theta; x), \tag{9}$$

where t is the number of tokens consumed, $\mathcal{L}_G$ is a mechanism-specific term (Section 4.3), and $\alpha(t)$ is the annealing schedule

$$\alpha(t) \;=\; \begin{cases} \alpha_{\max} \cdot t / t_w & t < t_w \\ \alpha_{\max} + (\alpha_{\min} - \alpha_{\max}) \cdot \dfrac{t - t_w}{t_d - t_w} & t_w \le t < t_d \\ \alpha_{\min} & t \ge t_d, \end{cases} \tag{10}$$

with $t_w = w_f T$ and $t_d = d_f T$ given as fractions of the budget T. A matched baseline/TMG pair shares the optimizer, data order, schedule, and seed, so differences are attributable to $\mathcal{L}_G$ and its schedule. The architectural route replaces $\mathcal{L}_G$ by a residual modification to f_θ at every position, with no schedule (Equations 14–15). Both routes read G at the current token with one batched GPU gather per step.

4.2 Efficiency metric

Because TMG supplies statistics the model would otherwise learn, we measure efficiency at fixed quality, which determines training cost. For a baseline trained for T tokens with final validation loss $\mathcal{L}_{\text{base}}(T)$ and a TMG run on the same schedule,

$$\textbf{tokens saved} \;=\; 1 - \frac{\min\{t : \mathcal{L}_{\text{TMG}}(t) \le \mathcal{L}_{\text{base}}(T)\}}{T}, \tag{11}$$

the fractional reduction in tokens needed to reach the baseline's final loss (undefined if it is not reached). **Wall-clock saved** is defined analogously with elapsed time and so includes per-step overhead. We also report the final paired loss difference

$$\textbf{gap} \;=\; \mathcal{L}_{\text{base}}(T) - \mathcal{L}_{\text{TMG}}(T), \tag{12}$$

which is defined even when tokens saved is not, with t-statistics for both.

4.3 Mechanisms

Mechanisms fall into two families; we also report two negative controls not expected to help (Appendix A.7).

Loss-based graph distillation. The base form is

$$\mathcal{L}_G(\theta; x) \;=\; \sum_i c(x_i) \cdot D_{\mathrm{KL}}\big(p(\cdot \mid x_i) \,\|\, P_\theta(\cdot \mid x_{\le i})\big), \tag{13}$$

where p is p_{raw} or p_{inv} and c is the graph confidence; all mechanisms below use p_{inv} (p_{raw} only in ablations, in no reported result). **Curvature-weighted distillation** (**curvature**) substitutes the bottleneck weight $b(x_i)$ for $c(x_i)$; a **learned-gate** variant replaces $\alpha(t)$ with a learned compatibility gate. **Multihop** uses p_{multihop} as the target; **freqweight** substitutes $w_{\mathrm{freq}}(x_i)$ for $c(x_i)$; **token-term** is Equation 13 with confidence weighting only. **Community** compares $Q(\pi(x_i), \cdot)$ with the model's induced community distribution, $P_\theta(\pi(x_i){=}b \mid x_{\le i}) = \sum_{v\,:\,\pi(v)=b} P_\theta(v \mid x_{\le i})$; **hub** applies cross-entropy against $\rho(x_i, x_{i-3:i-1})$ at matched positions. **Spectral-init** and **harp-init** use G only to initialize the input embeddings (spectral or HARP-style), with $\alpha \equiv 0$; **combined-overlap** sums the community, hub, multihop, and curvature terms. Community, hub, spectral-init, harp-init, combined-overlap, and token-term are evaluated on the 60B graph (Appendix A.6).

Architectural memory injection. Inspired by conditional-memory designs (Cheng et al., 2026), **engram** adds a residual at one intermediate layer with hidden state $h_i \in \mathbb{R}^d$. With $e_i = \sum_{v \in N_K(x_i)} p(v \mid x_i)\, E[v]$, the probability-weighted average of successor embeddings from f_θ's own embedding table, the module computes

$$g_i = \sigma\Big(\mathrm{RMSNorm}(h_i)^\top \,\mathrm{RMSNorm}(W_k e_i) \,/\, \sqrt{d}\Big), \qquad \tilde{v}_i = g_i \cdot W_v e_i, \tag{14}$$

$$h'_i = h_i + \mathrm{SiLU}\big(\mathrm{Conv1D}(\tilde{v}_{\le i})\big) + \tilde{v}_i, \tag{15}$$

with $\mathrm{Conv1D}$ a zero-initialized depthwise causal convolution and W_k, W_v learned projections; at initialization $h'_i = h_i + \tilde{v}_i$, which is not an identity map. **v1** uses single-hop e_i; **v2** additionally fuses a two-hop retrieval signal into e_i via a learned projection.

5 Experiments

All runs use a single-GPU pipeline with identical data order within each matched baseline/TMG pair. We report means over seeds with one-sample, two-sided t-statistics ($\alpha = 0.05$) on per-seed loss gaps and tokens saved; hyperparameters and hardware are in Appendices A.10 and A.2.

5.1 Experimental setup

Models. Gemma-2-style decoder-only Transformers at 172M, 416M, 671M, and 3B parameters (Appendix A.9).

Graphs. We build two independent graphs, from 60 billion and 350 billion tokens of the eight sources of the training mixture (FineWeb-Edu 25%, C4-English 14%, OpenWebText 10%, Wikipedia 7%, deduplicated Pile 9%, CodeParrot-clean 18%, OpenWebMath 12%, DCLM-Baseline 5%). As in practice, where the graph is built once from the full pretraining corpus from which every run draws, each graph covers more text than a single run consumes; since its statistics saturate (Section 3), what matters is only that the corpus is large enough for them to converge. The **350B graph** is the graph of record, used by every main-text result unless marked otherwise. The **60B graph**, built first, served for initial screening of all mechanisms (Appendix A.6) and as an independent replication (Section 5.7). Results are never pooled; Table 3 lists the graph, seeds, and reason for every experiment.

Budgets. The *reference budget* trains the 172M model for 100M tokens (13.2% of its Chinchilla-optimal budget), allowing 6–9 seeds per mechanism and all four sizes at one token count. The *Chinchilla-optimal budget* trains it for 756M tokens (20 per non-embedding parameter), the standard setting for pretraining from scratch.

5.2 Reference-budget comparison of mechanisms

Table 1 compares six mechanisms at the reference budget.[1]

[1]Five further mechanisms evaluated on the 60B graph and a single-seed token-term run are reported in Appendix A.6, and two negative controls in Appendix A.7.

Table 1: Reference budget (100M tokens, 172M), 350B graph; 6 seeds (9 for learned gate). "Reach": seeds reaching the baseline's final loss; t is on tokens saved over reaching seeds (d.o.f. 5; multihop 4). ‡From the 2 seeds with same-session baseline timing (-12.1%, -6.0%).

Mechanism	Tokens saved	t-stat	Reach	Wall-clock saved
Curvature	**14.2%**	**9.21**	6/6	-1.4%
Engram v2	10.8%	5.40	6/6	-4.6%
Engram v1	10.0%	4.47	6/6	-9.5%
Freqweight	10.0%	4.47	6/6	**+20.1%**
Multihop	10.0%	3.65	5/6	-9.2%
Curvature + learned gate	7.5%	3.50	6/9	$-9.1\%^{\ddagger}$

Curvature yields the largest reduction, 14.2% ($t = 9.21$), 31% more than the next mechanism (engram v2, 10.8%). All six mechanisms are significant at their achieved sample sizes; multihop reaches the threshold in 5 of 6 seeds ($t = 3.65$, d.o.f. $= 4$). Curvature with a learned gate varies most across seeds (6-seed run: $t = 2.32$, d.o.f. $= 3$); at 9 seeds, 6 reach the threshold ($t = 3.50$, d.o.f. $= 5$; loss gap over all 9 seeds $t(8) = 1.75$). On the 60B graph, combining four loss terms (combined-overlap) gives the largest reduction at the reference budget among mechanisms that reach the threshold in every seed, 16.7% ($t = 6.08$; Appendix A.6). At 172M, wall-clock saved is below tokens saved for every mechanism except freqweight (+20.1%), because a fixed per-step overhead is large relative to the smallest model's step time (Appendix A.12); from 416M upward curvature's wall-clock savings are positive (Section 5.4).

Curvature and freqweight share the target and differ only in the per-position weight (topological bottlenecks versus unigram frequency), so curvature's larger reduction (14.2% vs. 10.0%; $t = 9.21$ vs. 4.47) isolates the contribution of topological weighting.

Context-free target smoothing. Label smoothing and unigram-frequency smoothing ($\epsilon = 0.1$, 3 seeds each, same baseline seeds) increase validation loss in every seed (mean gap -0.0493, $t(2) = -7.23$; -0.0136, $t(2) = -2.09$) and never reach the baseline's final loss, whereas all six TMG mechanisms in Table 1 reach it in most or all seeds. The benefit thus comes from TMG's context-conditional, graph-weighted targets, not from smoothing.

5.3 Standard Chinchilla-optimal training

Pretraining from scratch is most commonly run to the compute-optimal token count. We train the 172M model for its full 756M-token Chinchilla-optimal budget with engram v1 (350B graph, 9 seeds). Engram v1 reaches the baseline's final loss with 8.1% fewer tokens ($t(8) = 7.61$) and 5.1% less calibrated wall-clock time ($t(8) = 4.56$), ending with a mean gap of $+0.0048$ ($t(8) = 3.05$). Every seed reaches the threshold, saving 4.1–14.0% of tokens and 0.8–11.0% of wall-clock time (Figure 3b; per-seed values and calibration in Appendix A.13). The advantage is not transient (Figure 3a): the mean gap is positive at every checkpoint from 50M tokens onward and holds at $+0.0047$ to $+0.0048$ from 550M tokens to the end, as training reaches the compute-optimal point. A 3-seed replication on the 60B graph agrees in sign and magnitude (mean gap $+0.0040$, $t(2) = 2.46$).

The two routes suit different budgets. The loss-based route distills toward a fixed first-order target and gives the largest savings below the compute-optimal point; over a full 756M-token run this static target no longer accelerates training (single seed; Appendix A.14). The architectural route supplies the graph through a learned gate, letting the model decide how much of it to use, and provides the compute-optimal gains above.

5.4 Model-size dependence

We train curvature at the reference budget at all four sizes with 3–6 seeds each; the 3B model uses the same 100M tokens, so all sizes are compared at an identical token count. Tokens saved and wall-clock saved both increase with model size, and the loss gap is significant at every size (Table 2, Figure 4). At 3B, curvature reaches the baseline's final loss after 80M of 100M tokens and stays ahead throughout the second half of training (Figure 4a). The 416M–671M plateau reflects the 5M-

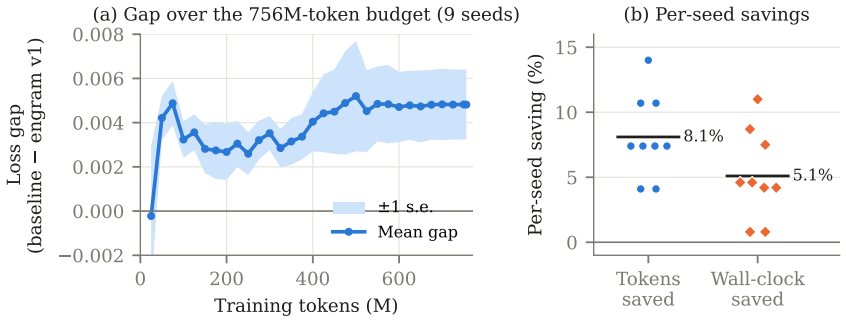


Figure 3: Engram v1, Chinchilla-optimal budget (172M, 756M tokens, 350B graph, 9 seeds). (a) Mean loss gap ±1 s.e.; positive from 50M tokens and stable to the end (final +0.0048, $t(8) = 3.05$). (b) Per-seed savings; bars are means (8.1%, $t(8) = 7.61$; 5.1%, $t(8) = 4.56$).

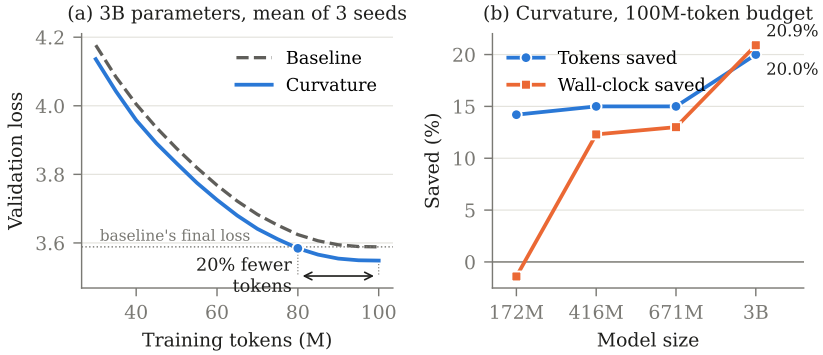


Figure 4: (a) Validation loss at 3B (3 seeds, 100M tokens); curvature reaches the baseline's final loss after 80M tokens. (b) Tokens and wall-clock saved versus model size (Table 2).

token checkpoint resolution; interpolating between checkpoints, tokens saved rises strictly with size: 14.2%, 15.9%, 17.2%, and 20.8%.

Wall-clock savings grow because curvature's per-step overhead is roughly constant in absolute terms (Appendix A.12) while the base model's step cost grows; at 3B, wall-clock saved (20.9%) matches tokens saved (20.0%), so the whole token reduction becomes a reduction in training time. At a fixed 100M tokens, larger models are also further from their compute-optimal budgets, so the trend reflects both factors.

5.5 Domain shift: code-only pretraining

Code makes up 18% of the graph's corpus. To test dependence on the natural-language-dominated training distribution, we train curvature and a baseline entirely on CodeParrot (172M, 100M tokens, 350B graph, 6 seeds). The effect persists: mean gap +0.0105 ($t(5) = 3.96$) and 12.5% tokens saved ($t(5) = 7.32$), with all 6 seeds positive and reaching the threshold; four save 15.0%, comparable to the 14.2% on the mixture (Appendix A.16). A graph built mostly from general text thus serves domain-specific pretraining, which suggests a route for domains with limited data: build the graph from abundant general text. Wall-clock saved is also positive (+2.9%, $t(5) = 1.04$), bounded at 172M by the per-step overhead that larger models amortize.

5.6 Downstream tasks

We evaluate zero-shot accuracy on eight tasks (HellaSwag, PIQA, ARC-Easy, ARC-Challenge, Winogrande, OpenBookQA, SciQ, LambadaOpenAI) at five checkpoints per seed over three seeds;

Table 2: Curvature, 100M tokens, 350B graph. Tokens saved uses 5M-token checkpoints; at 416M–3B all seeds reach it at the same one (zero variance). $^{\dagger}t$ of a variant interpolated between checkpoints (15.9%, 17.2%, 20.8%).

Model size	Seeds	Mean gap	t (gap)	Tokens saved (t)	Wall-clock saved
172M	6	$+0.0169$	4.83	14.2% (9.21)	-1.4%
416M	3	$+0.0151$	211.9	15.0% (239.7)†	$+12.3\%$
671M	3	$+0.0202$	61.9	15.0% (115.8)†	$+13.0\%$
3B	3	$+0.0400$	36.5	20.0% (124.9)†	$+20.9\%$

BoolQ is excluded for all conditions because of a task-specific effect of engram v2 (Appendix A.8). With equal task weights, the aggregate reaches the matched-accuracy threshold (20.0% tokens saved) in 2 of 3 seeds; the effect is carried by LambadaOpenAI, which improves in all three seeds, while the other tasks show no consistent direction (Appendix A.15). Equal weights give the 500-question OpenBookQA the weight of the 5,153-question LambadaOpenAI; weighting by question count, all three seeds reach the threshold (mean gap $+0.0052$, $t(2) = 4.42$), consistent with validation loss on the same runs ($t(2) = 7.42$), and we use this aggregate as the primary statistic. At 3B, LambadaOpenAI again improves significantly in all three seeds (net flips 88, 115, 89 of 5,153; $z = 2.92$, 3.68, 2.92) and SciQ in all three with smaller effects ($z = 2.74$, 0.58, 1.92); other tasks show no consistent effect. The gain thus appears where TMG's signal applies most directly: next-word cloze completion.

5.7 Robustness: learning rate, schedule, and graph instance

Baseline learning rate. Each baseline and its TMG counterpart share a fixed learning rate (3×10^{-4}), the standard design for isolating an auxiliary term. Because the baseline attains lower loss at 2–4× this rate (Appendix A.10), we repeated the 6-seed reference-budget comparison at 6×10^{-4}: curvature saves 13.3% of tokens ($t(5) = 12.65$, 6/6 seeds reaching the tuned baseline's final loss) versus 14.2% ($t(5) = 9.21$) at the default rate, so the gain is not due to an undertuned baseline.

Schedule. In a 6-point local sweep of curvature's schedule (single seed per setting), the default $\alpha_{\max}$ is a local optimum (2× smaller or larger values save 10.0% rather than 15.0%), while halving or doubling the warmup or decay-end fraction leaves tokens saved at 15.0% (Table 8), so no careful tuning is needed.

Graph instance and scale. The independently built 60B graph gives consistent results. Freqweight's final-loss gain is the same on both graphs (mean gap $+0.0088$ and $+0.0091$, positive in all 6 seeds on each); in a frequency-stratified analysis (single seed), curvature improves on rarer tokens with the larger graph, as expected when more co-occurrence data reduces estimation noise; and engram v1's compute-optimal gain replicates in sign and magnitude (Section 5.3). The conclusions do not depend on the graph instance.

Measurement pipeline. Two negative controls never reach the baseline's final loss in any of 6 seeds (Appendix A.7), so the pipeline does not register spurious gains.

6 Conclusion

Corpus token-transition statistics, supplied as a graph computed once from the corpus, accelerate language model pretraining across the budgets we tested. In Chinchilla-optimal training, engram v1 reached the baseline's final loss with 8.1% fewer tokens and 5.1% less wall-clock time, gaining in all 9 seeds ($t(8) = 7.61$, 4.56, 3.05 for tokens, time, and loss gap). At 100M tokens, curvature-weighted distillation, the strongest single mechanism, saved 14.2–20.0% of tokens from 172M to 3B parameters, with savings (up to 20.9% wall-clock) growing with size. The gains held under a tuned learning rate, across graphs, and on code ($t(5) = 3.96$), and appeared in next-word cloze completion. Built once per corpus and reused, the graph's cost is amortized over all training on that corpus, and fewer tokens-to-target lowers cost wherever that quantity matters: compute-optimal runs, data-constrained training, model development, and continued pretraining. Extensions (Appendix A.1) include combining the two routes and larger scale.

AI USE STATEMENT

Generative AI tools were used to help implementation and paper polishing.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. It involves no human subjects, user data, or personally identifiable information collected by the authors. All training and graph-construction data come from publicly released research corpora (FineWeb-Edu, C4, OpenWebText, Wikipedia, the deduplicated Pile, CodeParrot-clean, OpenWebMath, and DCLM-Baseline), used for research under their respective licenses. The models trained here are small research models that are not deployed. Like any language model trained on web text, they may reflect biases present in these corpora; TMG introduces no data source beyond the training corpus itself, and its frequency reweighting (Equation 3) down-weights rather than amplifies the most frequent continuations. The method's purpose is to reduce the tokens, compute, and energy needed to train language models; all experiments were run on a single workstation GPU, and the graph is built once on CPU and reused across runs, so we expect its net environmental effect to be a reduction in training cost.

REPRODUCIBILITY STATEMENT

All architecture dimensions, training hyperparameters, corpus mixture proportions, and graph-construction parameters are given in Section 3, Section 4, and Appendices A.9 and A.10. The efficiency metric (Section 4.2) can be recomputed from validation-loss trajectories alone. Appendix A.11 documents a validation-cache staleness failure mode encountered during this project and the epoch-matching safeguard adopted in response; the same failure mode can affect any resumable pretraining pipeline that logs validation metrics incrementally.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Nora Belrose, Quintin Pope, Lucia Quirke, Alex Mallen, and Xiaoli Fern. Neural networks learn statistics of increasing complexity. In *International Conference on Machine Learning (ICML)*, 2024.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning (ICML)*, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Xin Cheng, Rui Tian, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Chengqi Deng, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations (ICLR)*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2019.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *International Conference on Machine Learning (ICML)*, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

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

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations (ICLR)*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1995.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Huayang Li, Deng Cai, Jin Xu, and Taro Watanabe. Residual learning of neural text generation with n-gram language model. In *Findings of the Association for Computational Linguistics: EMNLP*, 2022.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. In *Conference on Language Modeling (COLM)*, 2024.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Timothy Nguyen. Understanding transformers via n-gram statistics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yann Ollivier. Ricci curvature of markov chains on metric spaces. *Journal of Functional Analysis*, 256(3):810–864, 2009.

Catherine Olsson, Nelson Elhage, Neel Nanda, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Aurko Roy, Rohan Anil, Guangda Lai, Benjamin Lee, Jeffrey Zhao, Shuyuan Zhang, Shibo Wang, Ye Zhang, Shen Wu, Rigel Swavely, et al. N-grammer: Augmenting transformers with latent n-grams. *arXiv preprint arXiv:2207.06366*, 2022.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. Beyond neural scaling laws: beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations (ICLR)*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Position: Will we run out of data? Limits of LLM scaling based on human-generated data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations (ICLR)*, 2022.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI Conference on Artificial Intelligence*, 2019.

Michihiro Yasunaga, Antoine Bosselut, Hongyu Ren, Xikun Zhang, Christopher D Manning, Percy Liang, and Jure Leskovec. Deep bidirectional language-knowledge graph pretraining. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

A Appendix

A.1 Limitations and future work

Our experiments span four model sizes from 172M to 3B parameters, two independently built graphs, natural-language and code-only training, and budgets up to and including the Chinchilla-optimal one. Within this range, curvature's token and wall-clock savings increase with model size (Section 5.4) and engram v1 is significant in standard compute-optimal training (Section 5.3). We note below where the present study ends and the extensions this suggests.

Graph construction. TMG requires one counting pass over the corpus before training: 27.9 minutes per billion tokens on a 4-core workstation, or about 7 days for the 350B-token graph. This cost is paid once per corpus and tokenizer and is independent of model size and of the number of training runs, so it is amortized over every run that reuses the graph; all experiments in this paper share two graphs. Counting parallelizes linearly with cores and corpus shards, so the build time falls in proportion to the hardware used. Because the graph saturates as the corpus grows (Section 3), it need not be rebuilt at ever larger scale: the 60B graph, built in about 1.2 days, already produces gains of the same sign as the 350B graph (Section 5.7). Incrementally updating the counts as new data arrive is a natural extension.

Larger scale. Given the upward trend of the savings with model size, the most direct extension is to larger models trained to their own compute-optimal budgets (48.3B tokens already at 3B parameters), a setting suited to multi-GPU training beyond the single-workstation setup used here (Appendix A.2).

Combining the two routes. The loss-based and architectural routes act through different pathways, the objective and the hidden state, and loss terms already combine on the 60B graph (combined-overlap, 16.7% tokens saved; Appendix A.6). Using distillation early in training together with the engram throughout is a natural way to combine the gains of both routes.

Community and hub structure on the largest graph. Community and hub routing were evaluated on the 60B graph, on which Louvain detection fits in a single GPU's memory (Appendix A.4); out-of-core or multi-GPU graph libraries would extend them to the 350B graph.

Continued pretraining and further domains. The code-only result shows that a single graph transfers across training distributions (Section 5.5); continued pretraining of existing checkpoints, low-resource domains, and multilingual corpora follow directly.

A.2 Hardware and software environment

All experiments were run on a single workstation GPU: one NVIDIA RTX PRO 6000 Blackwell Workstation Edition (97.9GB VRAM, driver 595.84, CUDA 12.8), paired with a 24-thread Intel Core Ultra 9 285K CPU and 122GB system RAM, running Ubuntu 26.04 LTS. Two logical CPU cores were excluded from all GPU-touching processes (`taskset`-pinned) after being identified as an intermittent hardware fault on this machine; this affects only CPU-side data-loading parallelism, not any reported numerical result. Partway through the compute-optimal replication (Appendix A.13), a system-level cooling regression on this same machine measurably raised sustained-load GPU throttling; this affects only wall-clock timing, not any token-indexed metric, and is corrected for via the calibration procedure described there. All training used a single GPU at a time; no multi-GPU or distributed training was used anywhere in this work.

Software: Python 3.14.4, PyTorch 2.11.0 (CUDA 12.8 build, cuDNN 9.19), Hugging Face Transformers 5.8.0, bf16 mixed-precision autocast, unfused AdamW. Training runs are driven programmatically (via `nbformat` 5.10.4 / `nbclient` 0.11.0) from a single source notebook shared across all conditions, so that a baseline and its matched TMG-assisted run differ only in the environment variables that toggle Equation 9's auxiliary term, never in code path. Downstream task evaluation uses `lm-evaluation-harness` 0.4.13.dev0, 0-shot, CPU-only, run independently of and without contending for the GPU with any training process.

A.3 Graph instance used in each experiment

Table 3 lists, for every experiment in the paper, the graph instance it uses, its number of seeds, and the reason for that choice. The 350B graph is the graph of record. The 60B graph was built first, and every experiment on it is either part of the initial mechanism screening or a cross-graph consistency check of a mechanism that was also run on the 350B graph.

Table 3: Graph instance used in each experiment. Seeds count TMG-assisted runs; each is paired with a baseline of the same seed. Budgets are the 100M-token reference budget unless marked Chinchilla-optimal (756M tokens at 172M parameters).

Experiment	Graph	Seeds	Reason for this graph
Mechanism comparison (Table 1)	350B	6 each (9 for learned gate)	Graph of record
Model size, 416M–3B (Table 2)	350B	3 per size	Graph of record
Engram v1, Chinchilla-optimal (Figure 3)	350B	9	Graph of record
Engram v2, Chinchilla-optimal	350B	3 (172M), 1 (416M)	Selects the engram variant for this budget
Curvature, Chinchilla-optimal (Table 12)	350B	1	Budget range of the loss-based route
Code-only training (Section 5.5)	350B	6	Graph of record, applied to a new training distribution
Downstream tasks, 172M (Section 5.6)	350B	3	Graph of record; the 3B evaluation reuses the model-size runs
Tuned baseline learning rate (Section 5.7)	350B	6	Graph of record
Schedule sweep (Table 8)	350B	1 per setting	Graph of record
Negative controls (Appendix A.7)	350B	6 each	Same graph as the comparison they control
Label and unigram smoothing (Section 5.2)	none	3 each	Graph-free baselines
Spectral-init, harp-init	60B	3 each	Initial screening; not significant at $n = 3$, not carried forward
Community, hub, combined-overlap	60B	3 each	Initial screening; their Louvain precomputation exceeds GPU memory at the 350B scale (Appendix A.4)
Token-term	60B	1	Diagnostic run during the initial screening
Curvature	60B	3	Cross-graph check; also on 350B
Freqweight	60B	6	Cross-graph check; also on 350B, same baseline seeds
Engram v1, Chinchilla-optimal	60B	3	Cross-graph check; also on 350B

A.4 Graph construction detail: 60B and 350B

Table 4 gives concrete construction statistics for both graph instances (Section 5.7). Graph construction is CPU-bound and parallelizes linearly with core count; the streaming bigram-counting pass required 27.9 minutes per billion tokens on a 4-core workstation (about 1.2 days for the 60B graph and 7 days for the 350B graph) and is performed once per corpus. Both are built over the same 262,144-token vocabulary from the eight sources described in Section 5.1; the 350B graph is a separate build at a larger corpus scale, not an extension of the 60B graph.

Table 4: Graph construction statistics for the two TMG instances used throughout this work. Entropy and confidence statistics are computed over the 255,314 tokens with outgoing edges in both graphs; the remaining 6,830 of the 262,144-token vocabulary have zero corpus frequency or zero observed successors in one or both graphs and are excluded from these two rows only (all other statistics and every downstream mechanism cover the full vocabulary). $H(u)$ here is the full-row successor entropy (Section 3), not the top-K-restricted entropy of p_{raw}, so it is not bounded by $\ln K$.

	60B graph	350B graph
Corpus tokens	60 billion	350 billion
Unique edges	433M	1.43B (3.3×)
On-disk size (CSR)	5.2GB	17.1GB
Mean successor entropy $H(u)$	3.866 nats	4.367 nats
Mean graph confidence $c(u)$	0.051	0.037
Vocabulary fraction with $c(u) > 0.1$	11.9%	8.3%

Construction is a single streaming pass over the corpus with pre-allocated arrays (entry counts known upfront from the source format's chunk metadata); serializing the accumulated counts of the 350B graph to CSR, after the counting pass, took 360.4 s with a peak resident memory of about 48GB. The total edge mass equals 99.88% of the corpus token count; the remainder corresponds to document and chunk boundaries, which produce no bigram edge. Because the graph is an aggregate count over the whole corpus and the training table keeps only six successors per token, it cannot store individual sequences: any held-out text contained in the corpus would contribute at most $2 \times 10^6 / 3.5 \times 10^{11} < 0.001\%$ of its counts.

Over the 255,314 shared-vocabulary tokens, the larger graph has higher mean successor entropy (4.367 vs. 3.866 nats), and entropy decreases for only 15.7% of these tokens from the 60B to the 350B graph. Additional data sharpens estimates for frequent tokens but also reveals successors of rarer tokens that were unobserved in the smaller corpus.

Saturation of the stored successor lists. Table 5 compares the top-K successor lists of the two graphs, binned by each token's count in the 350B-token corpus. Agreement increases steadily with frequency. The occurrence-weighted top-6 overlap is 0.80, and the three most frequent bins, which account for 98.6% of corpus occurrences, have a top-6 overlap of 0.71–0.82. Rare tokens, whose rows rest on few observations, change most between the two graphs. This matches the saturation account of Section 3 and the observation that curvature gains on rarer tokens with the larger graph (Section 5.7). The two corpora were drawn separately rather than by extending one into the other, so part of the residual difference at high frequency may reflect differences between the two samples rather than estimation noise.

Table 5: Agreement between the top-K successor lists ($K = 6$) of the 60B and 350B graphs, by token count in the 350B-token corpus, over the 255,314 tokens with successors in both graphs. "Share" is the bin's fraction of corpus occurrences; "Top-1" is the fraction of tokens with the same most probable successor; "Overlap" is the mean $|N_K^{60\mathrm{B}}(u) \cap N_K^{350\mathrm{B}}(u)|/K$; TV is the mean total variation distance between the two renormalized top-K distributions.

Token count	Tokens	Share	Top-1	Overlap	TV
$\geq 10^7$	3,041	73.5%	0.78	0.82	0.19
10^6–10^7	20,715	16.7%	0.67	0.75	0.26
10^5–10^6	95,882	8.4%	0.63	0.71	0.31
10^4–10^5	101,373	1.3%	0.54	0.63	0.39
10^3–10^4	25,554	0.04%	0.45	0.52	0.49
$< 10^3$	8,749	<0.01%	0.29	0.40	0.62
Occurrence-weighted	255,314	100%	0.72	0.80	0.21

Community detection (Louvain, Section 3.3) was run via GPU-accelerated cuGraph on the full 60B graph (all 433M edges), with a measured peak of 39.8GB VRAM; we did not attempt the analogous computation on the 350B graph, since a naive linear VRAM extrapolation from its 3.3× larger edge

count exceeds our GPU's 97.9GB. For this reason community and hub are reported only on the 60B graph (Appendix A.6).

A.5 Community structure and hub routing

Community structure. Token-level statistics are noisy at the tail, where most vocabulary entries have few observed successors. We obtain a much lower-variance, coarser signal by symmetrizing G's adjacency and running Louvain community detection (Blondel et al., 2008) on the result, giving a token-to-community assignment $\pi : V \to \{1, \ldots, C\}$ ($C = 1470$ communities; a mean of $\sim$180 tokens per community, though the distribution is skewed). Aggregating G's original *directed* edge weights by the community membership of their endpoints and row-normalizing gives a community-to-community transition matrix $Q \in [0,1]^{C \times C}$, where $Q(a, b)$ is the empirical probability that a token in community b immediately follows a token in community a in $\mathcal{D}$: each entry of Q aggregates over many token-level edges, trading token-level resolution for substantially lower variance.

Hub routing. $c(u)$ and κ derive from the first-order view of G, which is least informative at high-out-degree, high-entropy tokens such as punctuation, newlines, and discourse markers. We identify a hub set $\mathcal{H} \subset V$ of tokens in the top fraction of both out-degree and successor entropy and build a higher-order route table $\rho : \mathcal{H} \times V^3 \to V$ directly from $\mathcal{D}$: for a hub token and its three preceding tokens, ρ stores the single most frequently observed next token. Unlike every other property in this section, ρ is a point prediction rather than a distribution.

A.6 Secondary ablation on the 60B graph

Five of the mechanisms in Section 4.3 were evaluated only on the 60B graph (Table 6); community and hub, and therefore combined-overlap, depend on offline structures whose precomputation did not fit in GPU memory at the 350B scale (Appendix A.4). This exploratory comparison informed which mechanisms were carried forward to the 350B graph. Combined-overlap, which sums four loss terms, yields the largest reduction at the reference budget among mechanisms that reach the threshold in every seed (16.7%, $t = 6.08$), at a correspondingly higher per-step cost; this trade-off motivated our focus on single mechanisms. Spectral-init ($t = 3.00$) is close to the two-sided threshold at $n = 3$ (critical value 4.303, d.o.f. $= 2$); community, harp-init, and hub are not significant at this sample size and were not carried forward.

The remaining mechanism, token-term, distills toward the same target as curvature but weights positions only by graph confidence $c(x_i)$, without topological weighting. In a single-seed diagnostic run on the 60B graph at the reference budget, with its default schedule ($\alpha_{\max} = 1.0$), it produced no measurable change in validation loss (gap < 0.0001). Curvature, which adds topological weighting to the same target, improves on the baseline on both graphs (Table 1 on the 350B graph; 3 seeds on the 60B graph), which points to the topological weighting as the source of the effect.

Table 6: Reference-budget ablation, mechanisms restricted to the 60B graph, 3 seeds each (d.o.f.$=$ 2).

Mechanism	Tokens saved	t-stat	Reach	Wall-clock saved
Combined-overlap	16.7%	6.08	3/3	-152.0%
Spectral-init	13.3%	3.00	3/3	-36.4%
Harp-init	12.5%	0.38	2/3	-66.6%
Community	11.7%	1.48	3/3	-136.3%
Hub	20.0%	0.07	1/3	-220.4%

A.7 Negative controls

As a check that the pipeline does not register spurious gains, we instantiate $\mathcal{L}_G$ (Equation 9) with two variants not expected to help: **margin-gated** (gates the KL term on a heuristic combining low model confidence and high graph confidence, without a principled schedule) and **mask-replace** (a masked-replacement variant with no clear inductive-bias justification). As intended, neither reaches

Table 7: Base architectures. Chinchilla-optimal budget is $20\times$ non-embedding parameters, computed from unrounded parameter counts; the displayed non-embedding-parameter figures are rounded to two decimal places, so multiplying a displayed figure by 20 can differ from the displayed token budget by up to $\sim$0.1B (*e.g.* the 3B row: $2.42\text{B} \times 20 = 48.4\text{B}$ from the rounded figure versus the reported 48.3B from the unrounded one).

Size	Hidden	Layers	Heads	KV heads	Non-embed. params	Chinchilla tokens
172M	512	6	8	4	37.76M	756M
416M	800	13	16	4	206.38M	4.13B
671M	1024	17	16	4	392.24M	7.84B
3B	2048	32	16	8	2.42B	48.3B

the baseline's final loss in any of 6 seeds, confirming that the gains in Table 1 are not an artifact of the measurement pipeline.

A.8 BoolQ diagnosis for engram v2

This section documents a task-specific effect of the engram v2 variant that bears on the choice of injection point; it does not affect curvature. Engram v2's BoolQ accuracy remains at the majority-class level throughout training, while every other downstream task, and the baseline model's own BoolQ trajectory, show normal learning curves. We verified this is not an evaluation-harness artifact (engram v2's forward hook is not captured by a generic model-loading path; a corrected pipeline that reconstructs the hook explicitly gives an unchanged result), and found that curvature, despite also training against graph-successor structure, does not show the same collapse, ruling out a corpus-frequency-driven explanation common to any graph-trained mechanism. In our corpus the answer token "no" is $21.9\times$ more frequent than "yes" and has correspondingly higher graph degree; we hypothesize engram v2's direct, architectural injection of a graph-informed vector into an early hidden state is more readily absorbed into a persistent output bias than curvature's loss-only gradient signal, producing a bias that is severe for a binary-choice task but diluted across a typical multiple-choice task's 4–5 options. We could not fully confirm this mechanism and report it as the best-supported explanation rather than a settled one.

A.9 Base architectures

We instantiate f_θ as a Gemma-2-style (Gemma Team et al., 2024) decoder-only Transformer (Vaswani et al., 2017), with grouped-query attention (Ainslie et al., 2023) and rotary position embeddings (Su et al., 2021), following the pretraining paradigm established by Devlin et al. (2019) and Brown et al. (2020), using a 262,144-token vocabulary (a Gemma-2-style tokenizer), at four sizes shown in Table 7, spanning nearly two orders of magnitude in non-embedding parameter count that is used to define the Chinchilla-optimal token budget (Section 5.3).

A.10 Full hyperparameters

Optimizer and schedule. AdamW (unfused), weight decay 0.1, gradient clipping at global norm 1.0, cosine learning-rate schedule with a warmup fraction of 3% of total optimizer steps, base learning rate 3×10^{-4} unless swept (Table 9). Sequence length 512. Effective batch size held constant across model sizes via paired batch-size/gradient-accumulation settings, adjusted per model size only to stay within device memory (*e.g.* 3B: micro-batch 4, accumulation 8, or micro-batch 1, accumulation 32, both effective batch 32; the two settings gave indistinguishable optimization trajectories).

Graph-distillation loss terms. Default schedule for the curvature term: $\alpha_{\max} = 0.3$, warmup fraction 0.05, decay-end fraction 0.4, decaying linearly to a floor $\alpha_{\min} = 0$ thereafter; top-$K = 6$ successor candidates per token; count floor $\tau = 1000$. All other loss-based mechanisms (community, hub routing, multihop, freqweight) share this same $\alpha_{\max}/\alpha_{\min}$ warmup-and-decay schedule shape with their own mechanism-specific $\alpha_{\max}$, swept independently. The single token-term diagnostic run (Appendix A.6) used that term's own default schedule ($\alpha_{\max} = 1.0$, warmup fraction

0.1, decay-end fraction 0.8). Multi-hop propagation (Equation 7) uses $H = 3$ hops with decay $\beta = 0.5$. Frequency weight (Equation 8) uses exponent $\gamma = 0.5$ with floor $w_{\min} = 0.1$ and ceiling $w_{\max} = 10.0$. Table 8 gives the full local sensitivity sweep.

Table 8: Curvature schedule sensitivity at the reference budget (100M tokens, 172M parameters), 350B graph, single seed per setting, all against the same baseline. The default setting's 15.0% is a single run and differs from the 6-seed mean in Table 1 (14.2%) within seed-to-seed variance at 5M-token resolution.

Variant	Tokens saved	Notes vs. default (15.0%)
$\alpha_{\max}$ (default 0.3)	15.0%	–
$\alpha_{\max} = 0.15$ (low)	10.0%	underperforms default
$\alpha_{\max} = 0.60$ (high)	10.0%	underperforms default
Warmup 0.025 (short)	15.0%	no significant difference
Warmup 0.10 (long)	15.0%	no significant difference
Decay-end 0.2 (early)	15.0%	no significant difference
Decay-end 0.6 (late)	15.0%	no significant difference

Table 9: Baseline (no auxiliary term) learning-rate sensitivity at the reference budget: final validation loss relative to the default learning rate (3×10^{-4}), all other hyperparameters unchanged. Positive Δ indicates lower loss than the default-LR baseline. All comparisons use the default rate except the tuned-rate replication below.

LR (relative to default)	Δ vs. default-LR baseline	Trend
$0.5\times$	-0.333	
$1\times$ (default)	0.000 (reference)	–
$2\times$	$+0.145$	
$4\times$	$+0.146$	optimum range
$8\times$	-0.315	unstable

At this budget the baseline attains lower loss at 2–4$\times$ the default learning rate. A 6-seed replication of the curvature-versus-baseline comparison at $2\times$ the default (6×10^{-4}) gives a mean gap of $+0.0158$ ($t(5) = 16.39$), 13.3% tokens saved ($t(5) = 12.65$), and -1.8% wall-clock saved, within seed-to-seed variation of the default-rate results in Table 1 (14.2%, $t(5) = 9.21$, -1.4%); all 6 seeds reach the tuned baseline's final loss.

Architectural (engram) module. Hidden-size-matched key/value projections, zero-initialized depthwise causal convolution (kernel size 4), sigmoid compatibility gate computed on RMSNorm'd hidden state and retrieved-key inner product scaled by $\sqrt{d}$; v2 additionally fuses a two-hop retrieval signal via a learned linear projection before the gate. Injected via a forward hook at transformer layer index 1 (all model sizes).

A.11 Reproducibility safeguard: validation-cache staleness

An on-disk validation-token cache keyed only by a constant (the number of validation tokens), rather than jointly by condition and token budget, can be rebuilt (*e.g.* after an interrupted write) without invalidating validation losses already logged against the previous cache. A comparison between a checkpoint evaluated before such a rebuild and one evaluated after then compares two different validation sets and can produce a spurious effect; we observed a spurious loss-gap change of -0.07 from this cause during development. We key the training and validation caches independently and retrain any baseline that may predate a rebuild in the same execution as the condition it is compared against. All reported comparisons were checked for cache-epoch consistency under this procedure.

A.12 Wall-clock overhead

Wall-clock saved accounts for each mechanism's per-step cost in addition to its token reduction (Section 5.2). Curvature's overhead traces to CPU–GPU synchronization stalls (per-tensor `.item()`

calls) rather than arithmetic (45–73% wall-clock vs. under 0.3% FLOPs); batching these calls recovered most of the gap and is reflected in all curvature numbers reported in the main text. The remaining overhead is approximately constant per step and is therefore amortized at larger model sizes (Section 5.4). Engram v2, which we did not select for the compute-optimal budget, is profiled in Appendix A.13.

A.13 Engram v1 and v2: full Chinchilla-scale detail

Engram v1, 350B graph, full 9-seed detail. Table 10 gives the per-seed results for Section 5.3. The loss gap is positive and the baseline's final loss is reached in all 9 seeds; the mean gap ($+0.0048$, $t(8) = 3.05$), mean tokens saved (8.1%, $t(8) = 7.61$), and mean calibrated wall-clock saved ($+5.1\%$, $t(8) = 4.56$) are all significant.

Table 10: Engram v1, 172M parameters, 756M-token Chinchilla-optimal budget, 350B graph, all 9 seeds. Wall-clock saved is calibrated as described below.

Seed	Gap	Tokens saved	Wall-clock saved
0	$+0.0062$	10.7%	$+8.7\%$
1	$+0.0023$	7.4%	$+4.6\%$
2	$+0.0042$	7.4%	$+4.6\%$
3	$+0.0013$	4.1%	$+0.8\%$
4	$+0.0030$	7.4%	$+4.2\%$
5	$+0.0035$	7.4%	$+4.2\%$
6	$+0.0166$	14.0%	$+11.0\%$
7	$+0.0010$	4.1%	$+0.8\%$
8	$+0.0053$	10.7%	$+7.5\%$
Mean ($n = 9$)	**+0.0048**	**8.1%**	**+5.1%**
t**-stat**	**3.05**	**7.61**	**4.56**

Wall-clock calibration. Seeds 0–2 were trained before, and seeds 3–8 after, a system-level cooling regression on our workstation GPU (Appendix A.2). A controlled microbenchmark using the production training code at a reduced token budget reproduced the throughput ratio of seeds 0–2 at short duration and diverged only after about 20 minutes of continuous load, consistent with a thermal effect rather than a change in method, code, or data (the graph file and training code are identical across all 9 seeds). Because the effect influences elapsed time but not the token-indexed gap or tokens saved, we calibrate wall-clock saved for seeds 3–8 by applying the mean TMG-to-baseline throughput ratio of the three unaffected seeds (0.9716) to each seed's own baseline throughput, which separates engram v1's per-step overhead from the environmental effect. A single shared factor is used for all seeds; no per-seed fitting is involved. The token-indexed metrics are positive in every seed.

A 3-seed replication at 172M parameters on the independently constructed 60B graph (Section 5.7) gives a mean gap of $+0.0040$ ($t(2) = 2.46$), consistent in sign and magnitude with the 350B result; at $n = 3$ it is not significant on its own.

Engram v2. We report engram v2 at the compute-optimal budget because its comparison with v1 determined the variant used in that regime. At 172M parameters (756M-token Chinchilla-optimal budget, 3 seeds), gaps were $+0.0080$, -0.0002, $+0.0090$ (mean $+0.0029$, $t(2) = 1.12$), corresponding to 10.7%, never-reach, and 4.1% tokens saved respectively (2 of 3 seeds reach the threshold). At 416M parameters ($n = 1$, resumed from a partially completed run), the gap is $+0.0016$ with 4.3% tokens saved (threshold reached at 3950M of 4127.6M tokens), within the lower part of the 172M range. The simpler v1 variant is more consistent at this budget and has lower overhead (Table 11 profiles v2; Appendix A.8 documents a task-specific effect of v2), and is the one we use (Section 5.3).

The difference between projected ($+6.3\%$) and observed ($+21.9\%$) overhead is not explained by any operation isolated in these short-duration tests. Effects that appear only under sustained load (thermal throttling, memory fragmentation, storage I/O) are the most likely explanation; we did not instrument long runs to confirm this.

Table 11: Engram v2, 416M parameters, per-operation overhead profiling (isolated synthetic test, real production module code).

Component	Baseline	Engram v2	Overhead
Forward+backward (batch 8)	120.1 ms/step	128.5 ms/step	+7.0%
Forward+backward (batch 16)	238.5 ms/step	255.6 ms/step	+7.2%
Full optimizer step (batch 8)	514.9 ms	547.9 ms	+6.4%
One `evaluate()` call (1,957 examples)	13.03 s	13.19 s	+1.2%
One checkpoint save	3.20 s	3.28 s	+2.6%
Projected full-run total	36.40 h	38.71 h	+6.3%
Observed, full run	**33.83 h**	**41.23 h**	**+21.9%**

A.14 Operating regime of curvature-weighted distillation

This section records the budget range of the loss-based route. Below the compute-optimal budget, curvature-weighted distillation reduces tokens-to-target by 14.2–20.0% across all four model sizes (Tables 1 and 2). Table 12 traces the same mechanism when a 172M model is trained for its full 756M-token Chinchilla-optimal budget, with the schedule stretched to that budget. There, the fixed first-order distillation target no longer accelerates training, as the model has enough tokens to learn this structure unaided and to go beyond it (Section 3.2). This budget is served by the architectural route: at the same 756M-token budget, engram v1 reduces tokens-to-target by 8.1% and wall-clock time by 5.1%, with a positive saving in all 9 seeds (Section 5.3, Table 10).

Table 12: Budget range of the loss-based route: tokens saved by curvature-weighted distillation at checkpoints of a single run over the full 756M-token Chinchilla-optimal budget (172M parameters). Negative values indicate that the assisted run has not yet reached the baseline's loss at that point. At this budget the architectural route (engram v1) provides the gains (Table 10).

Tokens (of 756M budget)	Tokens-saved-to-date
25M	0.0%
100M	−12.9%
175M	−20.5%
300M	−14.1%
400M	−17.9%
550M	−20.4%
650M	−29.7%
756M (final)	−46.6%

A.15 Downstream-task per-seed breakdown

Table 13 gives the full per-task, per-seed z-score breakdown underlying the equally-weighted-aggregate diagnostic discussed in Section 5.6, and Table 14 gives the corresponding breakdown at 3B parameters. Both use z-scores from each condition's own combined standard error, $z = (\mathrm{acc}_{\mathrm{TMG}} - \mathrm{acc}_{\mathrm{base}})/\sqrt{\mathrm{se}^2_{\mathrm{TMG}} + \mathrm{se}^2_{\mathrm{base}}}$.

A.16 Code-domain per-seed breakdown

Table 15 gives the per-seed results underlying Section 5.5.

In the one seed whose full trajectory we inspected, curvature's advantage on code emerges after the first 30–35M tokens (the loss gap is slightly negative before that point) and persists for the rest of training. A possible explanation is that statistics from an 82%-natural-language corpus require a short adaptation period on code; we do not treat this single-trajectory observation as established.

Table 13: Per-task z-score at the final (100M-token) checkpoint, all 3 seeds, curvature, equally-weighted-aggregate diagnostic (see Section 5.6).

Task (n questions)	Seed 0 z	Seed 1 z	Seed 2 z
HellaSwag (10,042)	$+0.66$	$+0.40$	$+0.16$
PIQA (1,838)	$+0.20$	$+0.20$	-0.03
ARC-Easy (2,376)	-0.30	$+0.55$	$+0.76$
ARC-Challenge (1,172)	$+0.16$	-0.44	$+0.43$
Winogrande (1,267)	-0.44	$+0.12$	-0.16
OpenBookQA (500)	-0.37	-0.85	-0.38
SciQ (1,000)	$+1.17$	-0.49	-0.49
LambadaOpenAI (5,153)	$\mathbf{+4.55}$	$\mathbf{+2.50}$	$\mathbf{+2.65}$

Table 14: Per-task z-score at the final (100M-token) checkpoint, 3B parameters, curvature (see Section 5.6).

Task (n questions)	Seed 0 z	Seed 1 z	Seed 2 z
HellaSwag (10,042)	$+0.19$	$+0.51$	-0.34
PIQA (1,838)	$+0.33$	$+0.30$	-1.07
ARC-Easy (2,376)	$+1.77$	$+0.57$	$+0.86$
ARC-Challenge (1,172)	-0.05	$+0.05$	-1.12
Winogrande (1,267)	-0.87	$+0.04$	$+0.48$
OpenBookQA (500)	-0.48	$+0.00$	-0.09
SciQ (1,000)	$+2.74$	$+0.58$	$+1.92$
LambadaOpenAI (5,153)	$\mathbf{+2.92}$	$\mathbf{+3.68}$	$\mathbf{+2.92}$

Table 15: Code-only (100% CodeParrot) training, curvature, 172M parameters, 100M-token budget, per-seed results.

Seed	Gap	Tokens saved	Wall-clock saved
0	$+0.0097$	15.0%	$+12.0\%$
1	$+0.0016$	5.0%	-0.2%
2	$+0.0096$	15.0%	$+10.7\%$
3	$+0.0166$	15.0%	$+0.5\%$
4	$+0.0062$	10.0%	-5.5%
5	$+0.0190$	15.0%	$+0.1\%$
Mean ($n = 6$)	$+0.0105$	12.5%	$+2.9\%$
t-stat	3.96	7.32	1.04