THE COUNT: 
COUNTING N-GRAMS UNLOCKS SCALING ON HOST MEMORY AND TRAINING-FREE ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The scaling relationship of compute and performance for language models (LM) has been extensively studied; more recently, models are being designed to scale memory use, as well. In particular, adding input embeddings for n-grams scales memory separately from compute use, because only individual rows from the embedding table are read during each pass. However, existing methods train embeddings from scratch, which needs expensive GPU memory for fast reading and writing. Meanwhile, the ordinary memory of the host system is cheap and underutilized by these training methods. To scale on host memory, we introduce COUNT (Counting Offline Unlocks N-gram Transformers). COUNT scales n-gram representations while adding zero learned parameters by storing precomputed n-gram counts on host. We use the corpus frequencies to weight together the LM's own unembeddings of the next words most often following the current n-gram history. At a fixed compute budget, COUNT can be added as an additional input embedding for pretraining 1B-parameter LMs to improve validation loss as much as an LM with 58% more compute and task loss as much as 24% more compute. If GPU memory is limited, using COUNT to let backbone parameters fill GPU memory gives better performance than splitting memory with additional learned embeddings. Beyond efficiency, COUNT offers a new affordance: switching in a different table of n-gram counts from a domain-specific corpus at inference time to adapt to a new task without any training.

1 INTRODUCTION

Compute and memory are crucial resources for scaling language models (LMs). The tradeoff of compute and performance has been improved through optimal allocation of compute to parameters and training tokens (Kaplan et al., 2020; Hoffmann et al., 2022). Beyond compute, performance can also be bought through increased *memory*through architectures that scale sparse parameters, such as by activating only a subset of experts in mixture-of-experts (MoE) models (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022). The scaling of another type of sparsely activated parameters, input embeddings, has recently seen much exploration (Clark et al., 2022; Pagnoni et al., 2025; Huang et al., 2025; Liu et al., 2026; Cheng et al., 2026; Zhou et al., 2026). By embedding n-grams as well as individual tokens, many new parameters can be added to the embedding table while each lookup still only reads one row. Existing n-gram embedding work learns representations from scratch jointly with pretraining.

One consequence of learning n-gram embeddings from scratch is that they are not easily offloaded from expensive GPU memory during training, because their weights and optimizer state must be read and written quickly, increasing communication complexity. On the GPU, embeddings compete with the backbone model's parameters for memory. But a long line of language modeling research and settled knowledge has built representations of n-grams without expensive pretraining by instead counting the frequency of n-grams in a corpus (Jelinek & Mercer, 1980; Kneser & Ney, 1995; Brants et al., 2007). These methods refine the basic principle that the likelihood of a continuation can be estimated from counts of its occurrence after an n-gram history that is a suffix of the current context.

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

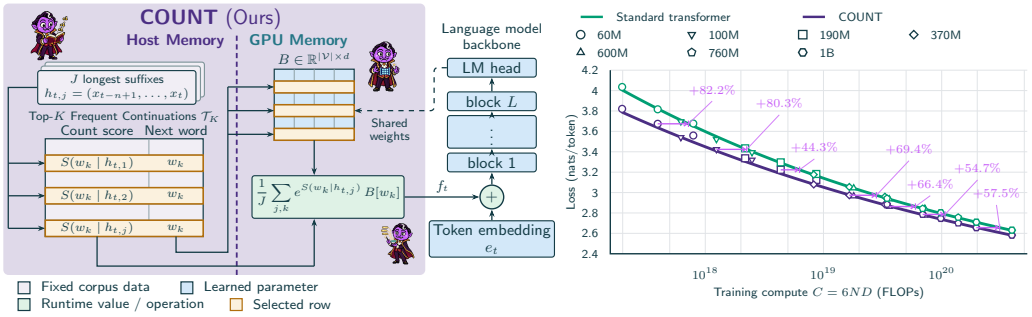

Figure 1: **Left:** COUNT uses corpus n-gram frequencies to score continuations and gather rows of the unembedding matrix into an additional input vector, f_t. The count score $S(w \mid h)$ can be stupid backoff (Brants et al., 2007), or any other frequency score based on $\mathrm{count}(h, w)/\mathrm{count}(h)$. Unlike n-gram embeddings learned from scratch, COUNT's additional storage can be offloaded to host RAM, even at training time. **Right:** COUNT reaches validation loss equivalent to a standard transformer trained with 44–82% more compute. It does this without increasing compute by instead scaling host RAM. Solid curves are the compute-optimal fits of each method's scaling law. Each violet guide shows the extra compute required to make a standard transformer reach the loss of an optimally trained COUNT run. Figure 2 shows task loss and downstream accuracy.

We propose *COUNT*, Counting Offline Unlocks N-gram Transformers, which represents n-grams by precomputed corpus next-word statistics stored in host RAM. For each n-gram history, corpus counts supply a fixed list of frequent next tokens. The backbone LM's unembedding head already learns representations of these predicted next words, so COUNT uses frequency scores to weight and combine unembeddings of next words as a new input representation summed with the token embedding. This adds no learned parameters, and since the large history table is read-only, it does not need to use GPU memory. Since COUNT's n-gram representations do not depend on expensive pretraining, they can also be cheaply recomputed over new corpora and updated at inference time with no additional parameter training.

In sum, our contributions are as follows:

- We introduce COUNT as a way to scale n-gram representations using host RAM at training, saving GPU RAM for model backbone parameters.
- By scaling host RAM to store 797 GB of n-gram statistics, COUNT improves LM and task loss across 60M–1B parameter backbones as much as if increasing compute 44–82% and 8–58%.
- COUNT achieves Pareto-optimal tradeoffs between loss and learned parameter count in comparisons with learned n-gram embeddings across backbone sizes from 60M–1B parameters.
- COUNT enables training-free adaptation by swapping in target-domain n-gram statistics at inference time for improvements in domain-specific evaluations in math, biomedicine, and law.
- We release[1] n-gram counts of the full Dolma 3 pretraining corpus (Olmo et al., 2026), alongside training code and pretrained checkpoints for COUNT and learned n-gram baselines.

2 Method

2.1 Preliminaries

A language model predicts the next token x_{t+1} from prefix $x_{1:t}$, each token in a vocabulary $\mathcal{V}$. Let d be the model's hidden width (i.e., size of each vocabulary item's embedding vector), $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ its token embedding matrix, and $e_t = E[x_t]$ the embedding of the token at position t. We write $h_t^{(0)} \in \mathbb{R}^d$ for the corresponding input to the first transformer block; conventionally, $h_t^{(0)} = e_t$.

[1]Release URL on publication

2.2 COUNT

COUNT (Figure 1) is a method for representing n-grams as an additional input to the LM backbone without adding any learned parameters. We use a precomputed, frozen table $\mathcal{T}_K$ that maps each of H stored n-gram histories to its K most frequent next words and their scores, derived cheaply from n-gram counts and approximating conditional log probabilities of next tokens. $K \ll |\mathcal{V}|$, the number of continuations retained per history, is chosen to retain just the head of the next word distribution, where the n-gram LM is more robust. Let $h_{t,j}$ be the longest stored suffix of $x_{1:t}$ at the j-th highest order of n. The lookup $\mathcal{T}_K[h_{t,j}]$ returns its top-K next words w_k and their count scores $S(w_k \mid h_{t,j})$, the log of the stupid-backoff estimate in Equation 2.

We use the LM's unembedding head as a basis $B \in \mathbb{R}^{|\mathcal{V}| \times d}$, whose rows we combine by the exponentiated scores to form the COUNT embedding f_t. Reusing this matrix avoids adding any learned parameters for COUNT.

$$\{(w_k,\, S(w_k \mid h_{t,j}))\}_{k=1}^{K} = \mathcal{T}_K[h_{t,j}], \qquad f_t = \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{K} \exp\big(S(w_k \mid h_{t,j})\big)\, B[w_k]. \tag{1}$$

We add the COUNT embedding and the token embedding e_t together as the input to the first transformer block, $h_t^{(0)} = f_t + e_t$.

2.3 N-GRAM COUNT-BASED SCORES

COUNT can make use of any count-based n-gram LM to score continuations. In our experiments we use *stupid backoff* (Brants et al., 2007) to score observed continuations by their count ratio and otherwise back off to a shorter history with a discount. All of this can be done offline, before pretraining, at a fraction of the cost of training a large LM on the same corpus.

Let $C(h, w)$ count occurrences of a history h followed by token w in the reference corpus, $C(h)$ count the history, and $C(w)$ count the token. The table stores every history h of the retained lengths with $C(h) \geq c_{\min}$, where $c_{\min}$ is the history-count threshold. *Stupid backoff* scores:

$$S(w \mid h) = \begin{cases} C(w)/\sum_{w' \in \mathcal{V}} C(w'), & h = \varnothing, \\ C(h,w)/C(h), & h \neq \varnothing,\ C(h,w) > 0, \\ 0.4\, S(w \mid h[1:]), & h \neq \varnothing,\ C(h,w) = 0. \end{cases} \tag{2}$$

Here $\varnothing$ is the empty history, whose score is the unigram frequency. The backoff scores $S(w \mid h)$ need not sum to one and are used without renormalization.

3 PRETRAINING EXPERIMENTS

3.1 BACKBONE LM

To represent a standard transformer backbone across a wide range of scales, we use configurations from the OLMo-core ladder up to 1B parameters and a token to parameter ratio of $D/N = 40$, a typical scale for pretraining development experiments. With these configurations we compare standard transformers, COUNT, or learned n-gram embeddings. We report training budgets as the token-to-parameter ratio D/N, where D is the number of training tokens and N counts backbone parameters excluding input embeddings but including the output head. Our primary comparisons use $D/N \in \{10, 20, 40\}$, rounded to full optimizer steps. Added n-gram parameters do not change these token budgets. We train the standard transformer at every size and budget with four random seeds; its reported points are seed means and its scaling-law fits use all four seeds' endpoints, whereas each COUNT and learned n-gram run is a single seed. We pretrain on `allenai/dolma3_mix-6T-1025-7B` and use the same corpus to obtain n-gram counts. Appendix A.1 gives the model ladder and optimization settings.

3.2 BUILDING N-GRAM TABLES

We count n-grams through order six over the full Dolma 3 (Olmo et al., 2026) pretraining corpus and construct continuation scores using the stupid-backoff procedure in Section 2.3. To vary table size,

we select which n-gram orders to store, prune low-count n-grams, discard histories below a history-count threshold, retain the top-K continuations, and quantize their scores. Since host RAM is inexpensive, our flagship experiments use the largest n-gram table which is sized to saturate the host RAM available on our compute nodes. That 797 GB table serves the 370M, 600M, 760M, and 1B backbones; the 60M, 100M, and 190M backbones use a 17.1 GB table built by the same procedure with tighter count thresholds, so COUNT's table capacity grows with backbone size across the ladder. See Appendix A.2 for all configurations of our primary table.

3.3 Learned n-gram embeddings

There are many approaches to learning n-gram embeddings in the literature; we compare COUNT with Over-Encoding (Huang et al., 2025) as implemented in LongCat (Liu et al., 2026) for its released codebase and models, demonstrated at large scales. Over-Encoding (Huang et al., 2025) augments token embeddings with learned representations of n-gram histories. Let N_{L} be the maximum learned n-gram order. Each n-gram order $n \in \{2, \ldots, N_{\mathrm{L}}\}$ uses R hash functions, indexed by r. Each hash selects a row from its own embedding table $T_{n,r} \in \mathbb{R}^{M_{n,r} \times d_s}$, where $M_{n,r}$ is the number of rows and d_s is the n-gram embedding width. Learned projections $P_{n,r} \in \mathbb{R}^{d \times d_s}$ map the retrieved embeddings to model width, giving the n-gram contribution q_t and additively updated transformer input:

$$q_t = \sum_{n=2}^{N_{\mathrm{L}}} \sum_{r=1}^{R} P_{n,r}\, T_{n,r} \left[\left(\sum_{j=0}^{n-1} x_{t-j} |\mathcal{V}|^j \right) \bmod M_{n,r} \right], \qquad h_t^{(0)} = e_t + q_t. \tag{3}$$

The tables and projections are learned jointly with the backbone.

LongCat (Liu et al., 2026) includes a technique called embedding amplification, which normalizes and scales the combined input, $\sqrt{d}\ \mathrm{LayerNorm}(h_t^{(0)})$. Here $\mathrm{LayerNorm}$ normalizes across the hidden features with learned gain and bias. We use bigram and trigram embeddings with four hash tables per order and table width $d/8$, projecting each retrieved vector to model width d. Following LongCat's guidance, we use a 1:1 ratio of learned embeddings to backbone parameters: input-token embeddings plus n-gram table parameters equal all remaining learned parameters, including the output head and projections.

We find that embedding amplification also works well with COUNT (Appendix B.5), so we apply it to both COUNT and learned n-gram embeddings. We also tried training standard transformers with embedding amplification and found it deteriorated performance at 1B, so we use no embedding amplification for this baseline in our comparisons with COUNT and learned n-gram embeddings.

3.4 Evaluation

We summarize OLMo-core's language-modeling and downstream evaluation suites with three aggregates.

LM validation loss is the mean token cross-entropy, in nats, across 11 validation sources based on Paloma (Magnusson et al., 2024): C4, Dolma Books, Dolma Common Crawl, Dolma peS2o, Dolma Reddit, Dolma Stack, Dolma Wiki, ICE, M2D2 S2ORC, Pile, and WikiText-103.

Task loss is the mean bits per byte (BPB) of reference answers from 74 downstream task configurations spanning knowledge and science, code, math and numerical reasoning, commonsense, reading and cloze, and symbolic and pattern reasoning. Likelihood metrics such as task loss give more stable comparisons of small base LMs than discrete metrics, which are noisier across model sizes and training budgets (Magnusson et al., 2025; Heineman et al., 2025), so task loss is our primary downstream metric.

Downstream accuracy is the mean accuracy across the 41 multiple-choice configurations among those tasks, using length-normalized accuracy (candidate log probability divided by character count) where available and ordinary accuracy otherwise.

Appendix A.3 specifies all the individual tasks and shows the task-category breakdown. Each source or task configuration receives equal weight.

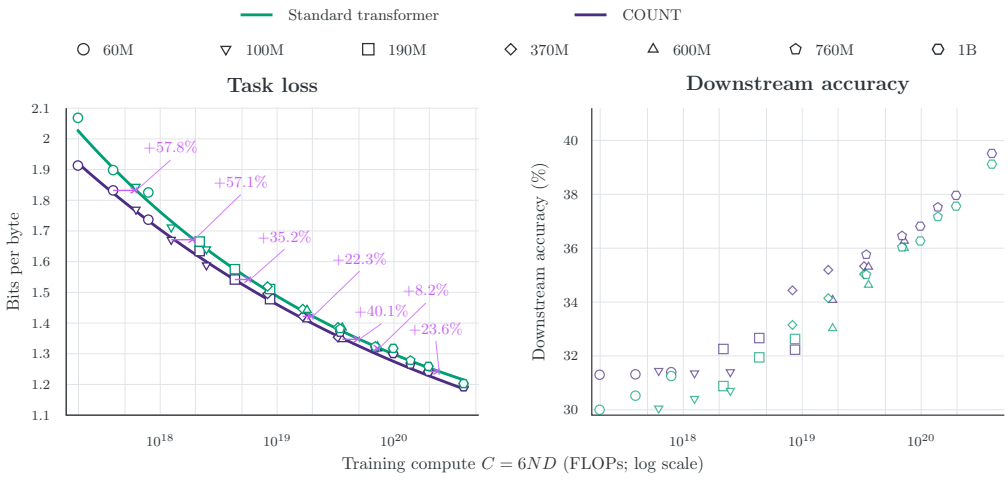

Figure 2: COUNT improves performance without increasing compute or learned parameters. By storing up to 797GB of n-gram count-based next-word scores, COUNT allows scaling on host RAM. **Left:** task loss against training compute at token to parameter ratios $D/N \in \{10, 20, 40\}$. The solid curves minimize fitted scaling laws for each arm, showing compute optimal performance (Hoffmann et al., 2022), and violet guides show the additional compute required to make a standard transformer reach the loss of an optimally trained COUNT run. **Right:** downstream accuracy for the same runs.

4 Pretraining results

4.1 Scaling host storage saves GPU compute

Figure 1 (right) compares LM loss, and Figure 2 task loss and downstream accuracy, for standard transformers and COUNT across 60M–1B parameter backbone LMs (COUNT uses the 17.1 GB table at 60M–190M and the 797 GB table from 370M up). COUNT improves downstream accuracy over standard transformer at matched compute at almost all budgets, by storing n-gram frequencies in host RAM.

We fit loss as a function of backbone size and training tokens separately for each method and loss metric, then select the fitted model/data allocation that minimizes loss at each compute budget (Hoffmann et al., 2022):

$$\widehat{L}(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}, \qquad \widehat{L}_{\text{opt}}(C) = \min_{N,D:\, 6ND=C} \widehat{L}(N, D). \tag{4}$$

Here E, A, B, α, and β are all fitted parameters. In our experiments, using COUNT improves LM loss as much as using 44–82% more compute. Meanwhile, task loss with COUNT improves comparably to a 8–58% increase in compute. Like other n-gram memory techniques, COUNT increases performance by scaling memory usage, but it is important to remember that unlike other methods COUNT only needs host memory, which is abundant in practice and largely uncontended by other parts of the language modeling pipeline.

4.2 Using host RAM for n-grams frees GPU RAM for backbone parameters

Figure 3 compares standard transformers, COUNT, and learned n-gram embeddings with 60M–1B backbones by total learned parameters. Learned parameters are the primary driver of additional GPU memory footprint, so this comparison allows hardware-agnostic assessment of optimality with respect to memory. The Pareto frontier follows COUNT, because its improvements add no learned parameters. Even where learned n-gram embeddings produce comparable performance, they do so at the cost of additional learned parameters to store learned embeddings, which cannot be easily offloaded during training and contend with backbone model parameters for GPU memory. Each column fixes D/N, so training tokens scale with the backbone rather than with added embedding parameters. The Pareto frontier connects configurations with the lowest measured loss, or highest

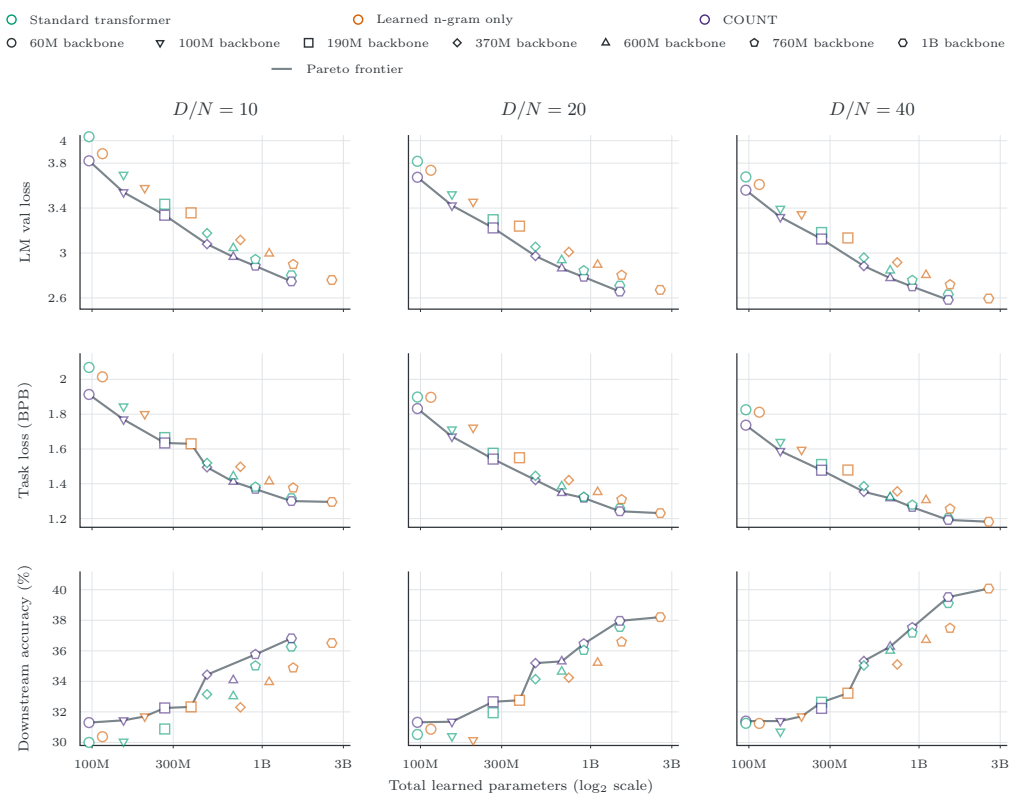

Figure 3: For a given budget of total learned parameters (a hardware-agnostic measure that drives GPU memory footprint) COUNT delivers best LM and task loss across a range of backbone model scales from 60M–1B, and downstream accuracy (bottom row) follows the same trend with more noise. Each column uses the indicated token to parameter ratio $D/N \in \{10, 20, 40\}$ relative to that run's own backbone.

accuracy, for their parameter budget. The bottom row of Figure 3 shows downstream accuracy on the same axes; it follows the loss rows with more noise, and the frontier occasionally passes through a learned n-gram embedding point.

5 Inference-time table swapping

Unlike learned n-gram embeddings, the n-gram tables used in COUNT all represent n-grams in the same geometry as scores over a vocabulary of next words. This allows the n-gram table to be modified at inference time without any training. We test two use cases where table swapping is desirable: whether a pretrained COUNT model can adapt to a new domain specific table or a smaller table for a tight deployed host-memory budget. Both experiments use COUNT pretrained on a 1B backbone at token-to-parameter ratios $D/N \in \{10, 20, 40\}$.

5.1 Adapting to a new domain without training

To test an inference time switch to a domain specific n-gram count table, we construct math, biomedical, and law tables of 83.398, 81.723, and 80.151 GB, respectively, compared with 797.487 GB for the original general-domain table. The math table uses 10.66 billion tokens from the Dolmino Math subset of the Dolma 3 Dolmino pool (Olmo et al., 2026). The biomedical and law tables use the health and crime-and-law topic splits of Dolma 3's olmOCR science-PDF source, containing 95.39 and 12.34 billion tokens. All use the pretraining tokenizer and n-grams through order six; Appendix A.2 gives the construction settings.

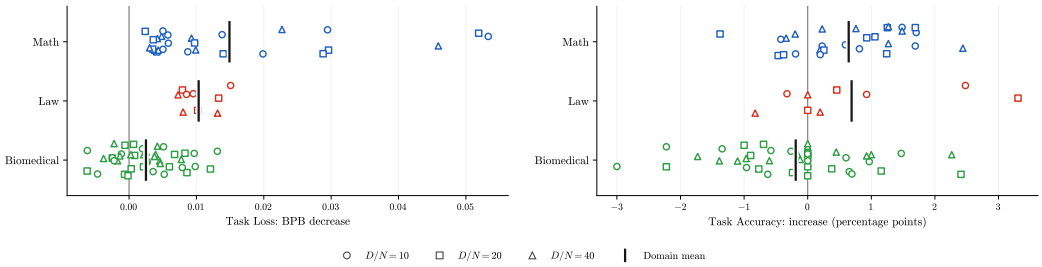

Figure 4: The count-based continuation scores in COUNT can be cheaply fit on new domains and swapped in at inference time with no training. We find in-domain task loss improves on all domains when interpolating 1:3 between a domain specific table and the domain general table used in pretraining this 1B parameter COUNT checkpoint. Accuracy results are more mixed but still increase on average for math and law domains. Each point is a result on one subtask at one training budget.

For each order, we combine the tables' candidate sets and interpolate their scores as $S_\lambda(w \mid h) = (1-\lambda)S_{\text{general}}(w \mid h) + \lambda S_{\text{domain}}(w \mid h)$, with zero contribution from a table when a candidate is absent from its row. We vary the domain-table weight $\lambda \in \{0, 0.25, 0.5, 1\}$; $\lambda = 0$ uses the original table and $\lambda = 1$ replaces it completely. We then select the eight highest-scoring candidates without renormalizing their scores. We measure task loss and task accuracy on matched domain evaluations: GSM8K, MATH, and Basic Skills arithmetic for math; MedQA, MedMCQA, LabBench, and ten MMLU biomedical subjects for biomedical; and the professional-law, international-law, and jurisprudence subsets of MMLU for law.

Figure 4 shows absolute changes in individual task results at $\lambda = 0.25$, the interpolation value that makes loss decrease for each domain. These are compared with each task's score with the original table at the same training budget. Mathematical and legal task losses and accuracy improve more than biomedical, which shows mixed results. For reference, the decrease in task loss we saw between COUNT with the generalist table and the standard transformer baseline (Figure 2) was 0.011-0.017 BPB, comparable to the improvements from moderate domain weighting. The full interpolation grid is in Appendix A.2.

5.2 Shrinking the table to fit a RAM-constrained deployment

The amount of host RAM at training time may be greater than what is available at deployment, so we ask how will performance degrade if we use a smaller n-gram count table at inference time. We replace the same model's original table with general-domain tables of 459.222, 142.037, 68.458, and 17.114 GB at each training budget. We evaluate only the original table and complete replacement ($\lambda \in \{0, 1\}$), loading one table at a time, on the full LM and task suites of Section 3.4. These tables vary in history-count and n-gram-count thresholds.

Figure 5 shows how much table storage can be removed while retaining better loss than a standard transformer. Starting with the 1B backbone COUNT checkpoints we change out the full sized 797GB n-gram table for more compressed tables and observe the host RAM-performance tradeoff. Notably, this can be done at inference time without any training, as the larger and smaller tables all represent n-gram histories in the same format as next word scores.

6 Training and inference throughput

While COUNT adds no learned parameters and can hide the host-resident table lookup time behind backbone LM GPU compute, we still need to measure its costs in practice. Adding no learned parameters means no added cost in the $C = 6ND$ sense and in parameter memory, but the sparse unembedding gather is real work that the estimate omits and GPU footprint depends on more than parameters, so we measure throughput and peak GPU memory (Table 10) directly. All of the following measurements use the 1B backbone parameter checkpoints of COUNT, learned n-gram embeddings, and standard transformer. We measure training throughput as median tokens per second (TPS) of optimization steps 50-100 on 8 H200 GPUs. For inference we measure prefill and decode

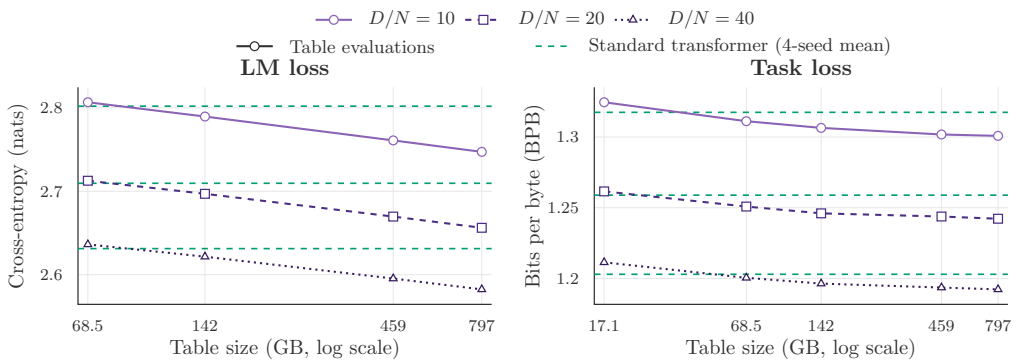

Figure 5: The n-gram table used in COUNT can be replaced at inference time with more compressed versions. This allows the method to work on host-RAM constrained deployments with a graceful and predictable performance degradation.

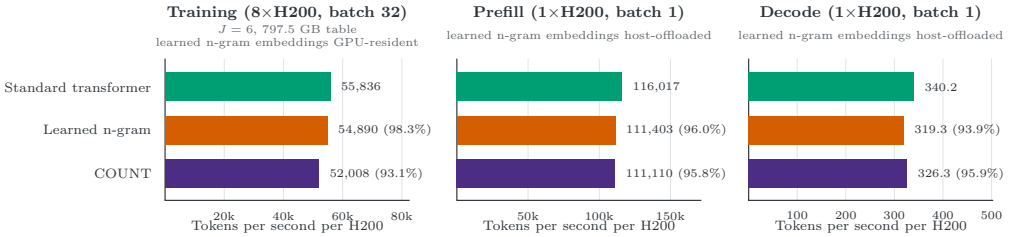

Figure 6: COUNT retains most of the training and inference throughput of a standard transformer. Its throughput is similar to learned n-gram embeddings, and further optimizations would not have far to go to recover both methods to full standard transformer throughput. Our headline configurations are measured here, and other configurations with smaller tables and lower J further improve COUNT's throughput, see Appendix B.1.

on one H200 after warm up batches. Figure 6 shows the headline configurations; Appendix A.4 gives the measurement configurations and Appendix B.1 reports every measured setting.

COUNT has a small impact on throughput at the 1B backbone scale, retaining 93.1–97.8% of standard transformer training throughput, 95.5–95.8% for prompt prefill, and 95.9–96.7% for autoregressive decoding. We use three optimizations to recover most of the standard transformer's throughput: (1) compiled, multithreaded host lookup; (2) prefetching lookups for training and prefill tokens while the GPU processes the preceding batch; and (3) a fused embedding reduction that gathers and sums candidate vectors without storing them separately.

We also optimize the learned n-gram embedding baseline that we compare against: (1) we implement sparse optimizer updates for only the retrieved embedding rows, avoiding a full-table gradient update at each step. At 190M with a 17.1 GB learned table, dense updates ran at 4.4% of standard transformer training throughput and sparse updates at 96.3%, and the 1B learned baseline in Table 7 trains at 98.3%. (2) We implement inference-time host offloading to gather learned embedding rows on the CPU and transfer them to the GPU, retaining 95.9–96.0% of standard transformer prefill throughput and 93.9–96.0% of decode throughput, similar to COUNT.

7 Related Work

7.1 N-grams in Neural Language Models

The past two years have seen much work on learning embeddings for n-grams, with the motivation that scaling embeddings increases performance predictably. The core problem all such work

grapples with is that independently representing all $|\mathcal{V}|^n$ possible n-grams is quickly intractable. One line of work uses multiple hashes of the n-grams to allow collisions that partially cancel out over different hash functions (Roy et al., 2022; Huang et al., 2025; Liu et al., 2026). Other work builds embeddings based on the tokens that it shares with other n-gram orders (Zhou et al., 2026) or combines token embeddings that make up the current n-gram suffix at inference time without any training (Qu et al., 2026). Yet another approach is to use an auxiliary model to embed the n-grams (Yu et al., 2025). Similarly, the backbone model's own hidden state can provide the contextual information, and additional embeddings are keyed just by the current unigram but further conditioned by the hidden state (Ding et al., 2026; Tseng & Sa, 2026). This approach can also be used with hashed n-gram embeddings by having the hidden state of the model gate the inclusion of the embeddings (Cheng et al., 2026).

Most similar to our work, Neubig & Dyer (2016) actually mix count-based next-word distributions with neural predictions conditioned on some count-based n-gram features. There is also more recent late-fusion-only work that interpolates the neural LM's output distribution with a nonparametric one at inference time, from nearest neighbors over a datastore of contexts (Khandelwal et al., 2020) or from unbounded-order n-gram counts (Liu et al., 2025b). In contrast to these approaches, our method can be seen as an early fusion combination of count-based and neural LMs where the count-based next word distribution is represented directly in the shared embedding space of the neural LM.

7.2 Tokenization and n-grams

A highly related thread of work explores n-gram tokens for a different motivation. They address the arbitrariness of token segmentation by either exploring ways to segment further and then represent combinations of tokens (Clark et al., 2022; Pagnoni et al., 2025) or by relaxing segmentation and allowing n-grams composed of many words to become single LM tokens (Liu et al., 2025a).

7.3 Retrieval-augmented language models

Though architecturally unrelated, retrieval-augmented LMs (Guu et al., 2020; Lewis et al., 2021; Borgeaud et al., 2022; Asai et al., 2023; Shao et al., 2024) share a motivation with COUNT: access to nonparametric information from a raw corpus can complement the parametric knowledge learned from the LM when it trains on the same data. Most notably, retrieval also provides the affordance of zero-training domain adaptation, which COUNT does as well. Where these methods practically differ from COUNT is that their nonparametric information is input directly as a token sequence into the LM, increasing context usage and compute. Also because COUNT stores only counts, its data store will always be much smaller than storing the raw corpus it is fit on.

8 Conclusion

High-bandwidth GPU memory is the scarcest and most expensive resource in current training and serving hardware, while host RAM is cheaper per byte and largely uncontended by the model itself. Learned n-gram embeddings buy their gains with parameters and optimizer state that must sit in GPU memory alongside the backbone. COUNT buys comparable gains with bytes on the cheaper tier, leaving GPU memory for backbone parameters. Representing n-grams by counts also opens possibilities beyond efficiency. A table of counts is an aggregate statistic rather than a corpus, so sharing it is often less sensitive and more compact than sharing the documents it summarizes, which offers a new way to ingest data that cannot be released as text. N-gram frequencies can also be estimated from far less data than pretraining requires. With only a small amount of domain specific data we fit count-based priors that steered a generalist, pretrained model to better in-domain performance. The long history of settled knowledge on count-based language modeling can likely produce useful next-word priors on even smaller corpora that are too small for parametric learning. Finally, because each row lists the next-token candidates and scores for a history, the prior a COUNT model consults can be read, corrected, or extended directly, by editing rows rather than by retraining.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods. We have not used generative AI tools for other tasks with required disclosure (design or provide feedback on research methodology or experiments, clean and reformat dataset, support qualitative and thematic data analysis, interpret results, propose or refine hypotheses), and the rest of the disclosure tasks (generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, assist with translation) are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, draft parts of a research paper, edit a research paper to improve readability, identify relevant literature, format references, and propose keywords for a research paper. We have reviewed all AI-assisted work. The paper was fully revised multiple times by all authors. The final method implementation was manually reviewed by the first author. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work aims to make language models more efficient in order to lower the cost and environmental impact of building and running them. By moving n-gram memory from GPU memory to host RAM and adding no learned parameters, COUNT lets smaller models reach quality that would otherwise require more compute or more expensive hardware, and its tables can be built from modest corpora. We hope this helps a wider range of people and organizations, including those with limited compute or limited data, to train and use capable models. COUNT also changes what a data contributor has to share: an n-gram count table records how often continuations follow short contexts, not the documents themselves, and is therefore less sensitive than raw pretraining text, although high-order counts over rare contexts can still reveal fragments of the source and should be released with the same care as other derived statistics. Our experiments use publicly available pretraining data and evaluation suites.

REPRODUCIBILITY STATEMENT

Section 2 defines COUNT and the stupid-backoff scoring it uses as well as the learned n-gram baseline in the same notation. Every pretraining run follows the OLMo-core ladder and recipe in Appendix A.1; the primary n-gram table's construction is stated in full in Appendix A.2, together with the smaller general-domain tables and the domain-specific tables in Table 3. Section 3.4 and Appendix A.3 specifies the evaluation suite, metrics, and aggregations, and Appendix A.4 records the hardware, batch sizes, and timing procedure behind every throughput number, with the complete measurements in Appendix B.1. Appendix B reports the additional results behind the ablations and inference-time sweeps. On publication we will release the n-gram counts extracted from the full Dolma 3 pretraining corpus, the table-building and training code, and pretrained checkpoints for COUNT and the learned n-gram baselines.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: learning to retrieve, generate, and critique through self-reflection, 2023. URL https://arxiv.org/abs/2310.11511.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6239.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego

de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2022. URL https://arxiv.org/abs/2112.04426.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In Jason Eisner (ed.), *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 858–867, Prague, Czech Republic, jun 2007. Association for Computational Linguistics. URL https://aclanthology.org/D07-1090/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Han Zhang, Yu-Kun Li, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4968–4990. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.226.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.365.

Ning Ding, Fangcheng Liu, Kyungrae Kim, Linji Hao, Kyeng-Hun Lee, Hyeonmok Ko, and Yehui Tang. MeKi: memory-based expert knowledge injection for efficient LLM scaling, 2026. URL https://arxiv.org/abs/2602.03359.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: a reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1246.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: a standard for language model evaluations. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5005–5033. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.282.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: retrieval-augmented language model pre-training, 2020. URL https://arxiv.org/abs/2002.08909.

David Heineman, Valentin Hofmann, Ian Magnusson, Yuling Gu, Noah Smith, Hanna Hajishirzi, Kyle Lo, and Jesse Dodge. Signal and noise: A framework for reducing uncertainty in language model evaluation. In *Advances in Neural Information Processing Systems 38*, NeurIPS 2025, pp. 19286–19327. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2025. doi: 10.52202/085713-0576.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset, 2021b. URL https://arxiv.org/abs/2103.03874.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego De Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George Van Den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems 35*, NeurIPS 2022, pp. 30016–30030. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-2176.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-tokenized transformer: Vocabulary is generally worth scaling, 2025. URL https://arxiv.org/abs/2501.16975.

F. Jelinek and R. L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice*, 1980.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? A large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, July 2021. ISSN 2076-3417. doi: 10.3390/app11146421.

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, 2020. URL https://arxiv.org/abs/2001.08361.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models, 2020. URL https://arxiv.org/abs/1911.00172.

R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pp. 181–184 vol.1, 1995. doi: 10.1109/ICASSP.1995.479394.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, Nov 2019. ISSN 2307-387X. doi: 10.1162/tacl_a_00276.

Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. LAB-Bench: measuring capabilities of language models for biology research, 2024. URL https://arxiv.org/abs/2407.10362.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: scaling giant models with conditional computation and automatic sharding, 2020. URL https://arxiv.org/abs/2006.16668.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems 35*, NeurIPS 2022, pp. 3843–3857. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-0278.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A. Smith, and Yejin Choi. SuperBPE: space travel for language models, 2025a. URL https://arxiv.org/abs/2503.13423.

Hong Liu, Jiaqi Zhang, Chao Wang, Xing Hu, Linkun Lyu, Jiaqi Sun, Xurui Yang, Bo Wang, Fengcun Li, Yulei Qian, Lingtong Si, Yerui Sun, Rumei Li, Peng Pei, Yuchen Xie, and Xunliang Cai. Scaling embeddings outperforms scaling experts in language models, 2026. URL https://arxiv.org/abs/2601.21204.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens, 2025b. URL https://arxiv.org/abs/2401.17377.

Ian Magnusson, Akshita Bhagia, Valentin Hofmann, Luca Soldaini, Ananya Jha, Oyvind Tafjord, Dustin Schwenk, Evan Walsh, Yanai Elazar, Kyle Lo, Dirk Groeneveld, Iz Beltagy, Hannaneh Hajishirzi, Noah Smith, Kyle Richardson, and Jesse Dodge. Paloma: A benchmark for evaluating language model fit. In *Advances in Neural Information Processing Systems 37*, NeurIPS 2024, pp. 64338–64376. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-2052.

Ian Magnusson, Nguyen Tai, Ben Bogin, David Heineman, Jena D. Hwang, Luca Soldaini, Akshita Bhagia, Jiacheng Liu, Dirk Groeneveld, Oyvind Tafjord, Noah A. Smith, Pang Wei Koh, and Jesse Dodge. DataDecide: how to predict best pretraining data with small experiments, 2025. URL https://arxiv.org/abs/2504.11393.

Graham Neubig and Chris Dyer. Generalizing and hybridizing count-based and neural language models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1163–1172. Association for Computational Linguistics, 2016. doi: 10.18653/v1/d16-1124.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal

Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2026. URL https://arxiv.org/abs/2512.13961.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.453.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. MedMCQA : A large-scale multi-subject multi-choice dataset for medical domain question answering, 2022. URL https://arxiv.org/abs/2203.14371.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534. Association for Computational Linguistics, 2016. doi: 10.18653/v1/p16-1144.

Yuwen Qu, Wenhui Dong, Chenyang Si, and Caifeng Shan. NGM: a plug-and-play training-free memory module for LLMs, 2026. URL https://arxiv.org/abs/2605.16893.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, 2016. doi: 10.18653/v1/d16-1264.

Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: a conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, Nov 2019. ISSN 2307-387X. doi: 10.1162/tacl_a_00266.

Aurko Roy, Rohan Anil, Guangda Lai, Benjamin Lee, Jeffrey Zhao, Shuyuan Zhang, Shibo Wang, Ye Zhang, Shen Wu, Rigel Swavely, Tao, Yu, Phuong Dao, Christopher Fifty, Zhifeng Chen, and Yonghui Wu. N-grammer: Augmenting transformers with latent n-grams, 2022. URL https://arxiv.org/abs/2207.06366.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: an adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, Apr 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6399.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. SocialIQA: commonsense reasoning about social interactions, 2019. URL https://arxiv.org/abs/1904.09728.

Rulin Shao, Jacqueline He, Akari Asai, Weijia Shi, Tim Dettmers, Sewon Min, Luke Zettlemoyer, and Pang Koh. Scaling retrieval-based language models with a trillion-token datastore. In *Advances in Neural Information Processing Systems 37*, NeurIPS 2024, pp. 91260–91299. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-2896.

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, 2017. URL https://arxiv.org/abs/1701.06538.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: a question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1421.

Albert Tseng and Christopher De Sa. L^3: Large lookup layers, 2026. URL https://arxiv.org/abs/2601.21461.

David Wadden, Kejian Shi, Jacob Morrison, Alan Li, Aakanksha Naik, Shruti Singh, Nitzan Barzilay, Kyle Lo, Tom Hope, Luca Soldaini, Shannon Zejiang Shen, Doug Downey, Hannaneh Hajishirzi, and Arman Cohan. SciRIFF: a resource to enhance language model instruction-following over scientific literature. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6083–6120. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.310.

Alexander Wettig, Kyle Lo, Sewon Min, Hannaneh Hajishirzi, Danqi Chen, and Luca Soldaini. Organize the web: Constructing domains enhances pre-training data curation, 2025. URL https://arxiv.org/abs/2502.10341.

Da Yu, Edith Cohen, Badih Ghazi, Yangsibo Huang, Pritish Kamath, Ravi Kumar, Daogao Liu, and Chiyuan Zhang. Scaling embedding layers in language models. In *Advances in Neural Information Processing Systems 38*, NeurIPS 2025, pp. 36060–36086. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2025. doi: 10.52202/085713-1074.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1472.

Ziyin Zhang, Zhaokun Jiang, Lizhen Xu, Hongkun Hao, and Rui Wang. Multiple-choice questions are efficient and robust LLM evaluators, 2024. URL https://arxiv.org/abs/2405.11966.

Wuyang Zhou, Yuxuan Gu, Giorgos Iacovides, Yuning Qiu, Qibin Zhao, and Danilo Mandic. Tensorizing engram: Sharing latents across n-gram embeddings is beneficial in LLMs, 2026. URL https://arxiv.org/abs/2606.08347.

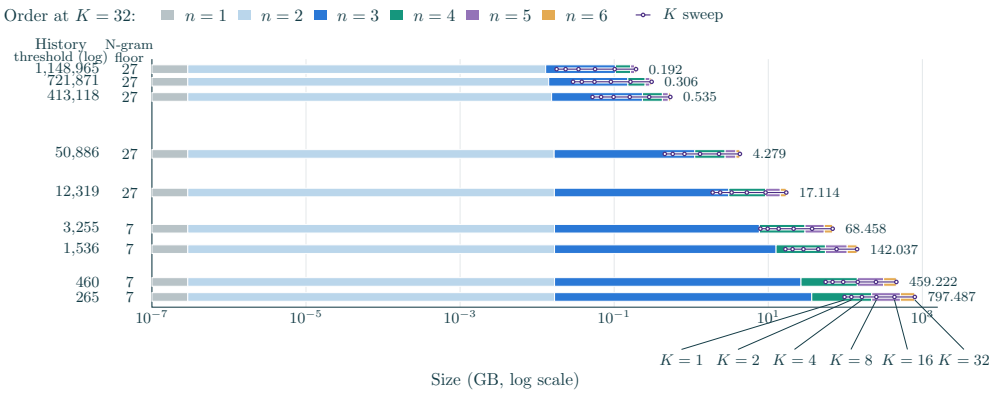

Figure 7: The size of our n-gram tables depends on several configurations. History and n-gram occurrence thresholds limit the number of rows in the table. Each order of n-gram is its own set of rows in the table, with the second order accounting for most of the rows. Finally lowering K to store fewer top continuations reduces the table width.

A Experimental details

A.1 OLMo-core

Table 1 lists the backbone geometries and Table 2 the training recipe; ladder labels are nominal sizes before any n-gram memory is added.

Table 1: OLMo-core model ladder configurations. The feed-forward width is the intermediate width of each transformer block.

Ladder label	Layers	Hidden width	Heads	Feed-forward width
60M	8	384	8	1,536
100M	12	512	8	2,048
190M	12	768	12	3,072
370M	16	1,024	16	4,096
600M	16	1,280	16	5,120
760M	16	1,536	16	6,144
1B	16	2,048	16	8,192

A.2 N-gram tables

Our primary general-domain table stores the top $K = 32$ continuations for every history of order one through six seen at least 265 times, after dropping order-4 to order-6 n-grams with fewer than seven occurrences, and occupies 797.487 GB.

Figure 7 shows how the general-domain table sizes depends on the n-gram order, the history-count threshold, the per-n-gram count floor, and K; sizes are decimal gigabytes including headers. Stored base-10 log scores are quantized to 256 levels per order, spanning the smallest retained finite score through zero, and converted to natural-log scores at read time.

The domain tables in Table 3 are built from the Dolmino Math subset of Dolma 3 (10.66B tokens) and from the Health and Crime and Law topics of Dolma 3's olmOCR science-PDF source (95.39B and 12.34B tokens), following the WebOrganizer topic categories (Wettig et al., 2025; Olmo et al., 2026). All use the pretraining tokenizer, orders through six, a count floor of one, and a backoff factor of 0.4; the biomedical and law history-count thresholds are the lowest integers that fit within the math table's 83.398 GB.

Table 2: Training configuration. P is the backbone parameter count excluding input embeddings but including the output unembedding; added n-gram parameters do not set the training budget.

Setting	Value
Tokenizer	`allenai/dolma2-tokenizer`
Vocabulary	100,278 tokens; embedding allocation padded to 100,352 rows
Sequence length	8,192 tokens
Position encoding	Rotary position embeddings with base 500,000
Precision	Bfloat16 model computation; float32 gradient reduction
Optimizer	AdamW with anomalous-update skipping (`SkipStepAdamW`); $(\beta_1, \beta_2) = (0.9, 0.95)$ at the listed batch sizes; $\epsilon = 10^{-8}$
Weight decay	0.1; input embeddings, embedding-amplification parameters, and learned n-gram tables and projections are exempt
Peak learning rate	$0.00235(P/108{,}000{,}000)^{-1/3}$
Schedule	Warmup over P tokens, followed by a stable learning rate and linear decay to zero over the final 10% of each successive reporting interval (WSD-S)
Token budgets	Reporting endpoints at $D/N = 10$, 20, and 40, rounded to full optimizer steps
Global batch	The OLMo-core ladder targets $2048 \cdot 160\,(N/10^8)^{2/3}$ tokens per step, where N counts non-embedding parameters, and rounds the target to the nearest multiple of the microbatch size times the number of GPUs times the 8,192-token sequence length. The microbatch was set per run to fit the heterogeneous GPUs available at the time, so runs at the same size can differ in their rounded batch; token budgets are identical and only step counts differ
Random seed	1042 for every COUNT, learned n-gram embedding, and amplified standard transformer run; standard transformer baselines are means over seeds 1, 2, 3, and 42
COUNT	Unembedding matrix B as the next word basis; tables store $K = 32$ candidates per history and training uses $K = 8$ per order over orders one through six with embedding amplification; 17.1 GB table at 60M–190M and 797.5 GB at 370M–1B
Learned n-gram embeddings	Orders two and three, four hash tables per order, embedding width $d/8$ projected to d; rows set by the 1:1 ratio: 0.40M rows across the eight tables at 60M, 0.79M at 100M, 1.19M at 190M, 2.11M at 370M, 2.63M at 600M, 3.16M at 760M, and 4.21M at 1B (0.04–2.16 GB of bfloat16 table weights)
Learned-table updates	Sparse row-wise AdamW on retrieved rows, using the backbone learning-rate schedule and zero weight decay

Table 3: Domain-specific tables used in Figure 4. Sizes are decimal GB with 8-bit scores; the source candidate width and intermediate history floor are the intermediate-construction settings before packing the final top-K rows.

Domain	Size (GB)	K	History-count threshold	Source candidate width	Intermediate history floor
Math	83.398	32	4	256	4
Biomedical	81.723	32	45	256	32
Law	80.151	32	8	256	6

A.3 Evaluation breakdown

The in-loop suite is `ai2-olmo-eval` 0.9.0. LM validation loss averages token cross-entropy within each of the 11 sources before averaging sources. Task loss averages the 74 per-configuration BPB values, which include reference-answer scoring for code, math and open-ended tasks that report no accuracy; the accuracy aggregate covers the 41 multiple-choice configurations. Figure 8 groups the configurations into task categories; subject subsets, splits, prompt formats and language variants count separately, and every configuration keeps equal weight. The configurations derive from the following parent benchmarks.

- **Knowledge and science:** ARC-Easy and ARC-Challenge (Clark et al., 2018); MMLU (Hendrycks et al., 2021a), as its four subject groups on the validation and test splits; LAB-Bench DbQA and ProtocolQA (Laurent et al., 2024); MedQA (Jin et al., 2021); MedMCQA (Pal et al., 2022); Natural Questions (Kwiatkowski et al., 2019); Jeopardy (Gu et al., 2025); Basic Skills common knowledge (Olmo et al., 2026).
- **Code:** HumanEval (Chen et al., 2021); MBPP (Austin et al., 2021); MT-MBPP, machine-translated MBPP in 17 further programming languages (Olmo et al., 2026); Basic Skills coding (Olmo et al., 2026).
- **Math and numerical reasoning:** GSM8K (Cobbe et al., 2021); MATH (Hendrycks et al., 2021b) in the Minerva format (Lewkowycz et al., 2022), as seven subject subsets; DROP (Dua et al., 2019); Basic Skills arithmetic (Olmo et al., 2026). GSM8K and MATH accuracy in Figure 4 uses the four-choice versions of these test sets, whose distractor answers were produced by Zhang et al. (2024); the model ranks the answer texts as for every other accuracy task.
- **Commonsense:** CommonsenseQA (Talmor et al., 2019); HellaSwag (Zellers et al., 2019); PIQA (Bisk et al., 2020); SocialIQA (Sap et al., 2019); WinoGrande (Sakaguchi et al., 2020).
- **Reading and cloze:** CoQA (Reddy et al., 2019); LAMBADA (Paperno et al., 2016); SQuAD (Rajpurkar et al., 2016); QASPER yes/no (Dasigi et al., 2021); SciRIFF yes/no (Wadden et al., 2025).
- **Symbolic and pattern reasoning:** Basic Skills logical reasoning, pattern recognition and string operations (Olmo et al., 2026); CopyColors (Gu et al., 2025).

All tasks use the OLMES implementations (Gu et al., 2025). ARC, MMLU, CommonsenseQA, PIQA and SocialIQA each contribute two configurations, one in the ranked-choice (`mc`) prompt format and one in the completion (`rc`) format; every other benchmark contributes its completion format only.

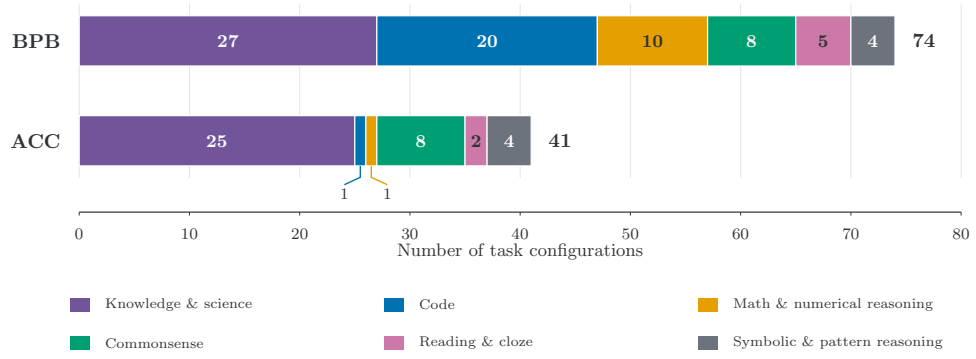

Figure 8: Composition of the task-loss (BPB) and accuracy (ACC) aggregates by qualitative task category. Each eligible configuration contributes once: 74 configurations for BPB and 41 for accuracy. Subject subsets, splits, prompt formats, and language variants are counted separately. Categories describe task families rather than isolated skills.

A.4 THROUGHPUT CONFIGURATIONS

All throughput measurements use the 1B ladder configuration, include lookup and host–GPU transfer in the timed workload, use the 1:1 learned-embedding ratio, keep COUNT tables in host RAM, and (for inference) use $D/N = 40$ checkpoints on C4 validation prompts. Compiled lookup searches sorted histories with multiple CPU threads; lookup caching reuses retrieved results; prefetch prepares lookups while the GPU processes the preceding batch; fused embedding combines row gathering and weighted summation; sparse updates optimize only retrieved learned rows; host offloading keeps learned tables on the CPU with their projections on the GPU; KV caching and CUDA graphs are the usual inference optimizations. Tables 4–6 give the settings for each workload.

Table 4: Training TPS experiment configuration.

Setting	Value
Hardware	Eight H200 GPUs
Batch per GPU	Four sequences; six gradient-accumulation steps
COUNT table	17.1 or 797.5 GB
COUNT selection	$K = 8$; $J = 6$ or 1
Learned table	GPU-resident
Optimizations	Prefetch; fused embedding; sparse updates
Runtime	FlashAttention 3; block/head compilation; fully sharded data parallelism
Timing	100 steps; median of six TPS samples at steps 50–100

Table 5: Prefill TPS experiment configuration.

Setting	Value
Hardware	One H200 GPU
Batch sizes	1 and 8 sequences
Prompt length	8,192 tokens
COUNT table	797.5 GB; $K = 8$ per order, orders 1–6
Learned table	2.008 GiB; GPU-resident or host offloading
Optimizations	Compiled lookup; lookup caching; prefetch; fused embedding; KV caching
Lookup workers	Eight workers; ten batches of lookahead
Runtime	PyTorch attention
Timing	Three warmup batches; 96 timed batches at batch 1 or 12 at batch 8

Table 6: Decode TPS experiment configuration. Prompt processing and graph capture are excluded from the timed generation steps.

Setting	Value
Hardware	One H200 GPU
Batch sizes	1 and 256 sequences
Prompt / generation	512 prompt tokens; 256 greedily generated tokens
COUNT table	797.5 GB; $K = 8$ per order, orders 1–6
Learned table	2.008 GiB; GPU-resident or host offloading
Optimizations	Compiled lookup; lookup caching; fused embedding; KV caching; CUDA graphs
Lookup threads	Eight
Runtime	PyTorch attention
Timing	Ten warmup steps; 246 timed generation steps

B Additional results

B.1 Full throughput results

Tables 7–9 report every measured setting behind Figure 6. Rates are tokens per second per executing H200; parentheses give each treatment's share of the standard transformer. Table 10 reports the peak GPU memory of the same measurements. The learned baseline's training rate is a single measurement, and COUNT was not measured in the separate host-offloading comparison.

Table 7: Training throughput at batch 32 on eight H200s by COUNT lookup (J) and table size.

	$J = 6$ 17.1 GB	$J = 1$ 17.1 GB	$J = 6$ 797.5 GB	$J = 1$ 797.5 GB
Standard transformer	55,836	55,836	55,836	55,836
Learned n-gram	54,890 (98.3%)	54,890 (98.3%)	54,890 (98.3%)	54,890 (98.3%)
COUNT	52,127 (93.4%)	54,585 (97.8%)	52,008 (93.1%)	54,532 (97.7%)

Table 8: Prefill throughput on one H200 by batch size and learned-embedding placement.

	Batch 1 GPU-resident	Batch 8 GPU-resident	Batch 1 host-offloaded	Batch 8 host-offloaded
Standard transformer	116,017	122,389	116,017	122,389
Learned n-gram	115,181 (99.3%)	122,091 (99.8%)	111,403 (96.0%)	117,402 (95.9%)
COUNT	111,110 (95.8%)	116,931 (95.5%)	—	—

Table 9: Decode throughput on one H200 with CUDA graphs by batch size and learned-embedding placement.

	Batch 1 GPU-resident	Batch 256 GPU-resident	Batch 1 host-offloaded	Batch 256 host-offloaded
Standard transformer	340.2	19,216	340.2	19,216
Learned n-gram	290.8 (85.5%)	18,795 (97.8%)	319.3 (93.9%)	18,448 (96.0%)
COUNT	326.3 (95.9%)	18,586 (96.7%)	—	—

B.2 Table swap interpolation on loss

Figure 9 shows the absolute task-loss changes at all four interpolation weights, averaging subjects within MATH, LabBench and the MMLU biomedical group.

B.3 Inference-time K and J variation

Starting from the trained 1B COUNT model, we test its sensitivity to two inference-time settings without retraining: the number of candidates kept per order, K, and the number of longest matching orders averaged, J (Figures 10 and 11). Both show little tolerance: only the smallest departures from the trained setting remain above the standard transformer, and larger changes in either direction lose most of the improvement. A sensitivity sweep that pretrains each configuration would likely retain more of the improvement, but we forgo these additional expensive pretraining experiments. We did pretrain two ablations at 190M: the maximum n-gram order (Appendix B.4) and embedding amplification (Appendix B.5).

B.4 Training with fewer n-gram orders

Figure 12 compares 190M COUNT trained with orders one through three against orders one through six, with the same table and K. Six orders give lower LM and task loss at every budget, so we train with all six.

Table 10: Peak GPU memory in GiB per executing GPU for each throughput measurement, as allocated (reserved) high-water marks of the PyTorch allocator. Training is sharded across eight H200s; inference runs on one H200. The learned n-gram baseline's GPU-resident table holds 2.16 GB.

Workload	Batch	Standard transformer	Learned n-gram	COUNT
GPU-resident learned embeddings				
Training ($J = 6$, 17.1 GB)	32	60.8 (61.7)	63.4 (64.6)	61.0 (62.0)
Training ($J = 1$, 17.1 GB)	32	60.8 (61.7)	63.4 (64.6)	60.9 (62.0)
Training ($J = 6$, 797.5 GB)	32	60.8 (61.7)	63.4 (64.6)	61.0 (62.0)
Training ($J = 1$, 797.5 GB)	32	60.8 (61.7)	63.4 (64.6)	60.9 (62.0)
Prefill	1	4.4 (6.1)	6.4 (12.2)	4.4 (6.1)
Prefill	8	14.4 (20.1)	16.4 (21.9)	14.4 (18.9)
Decode	1	4.0 (4.9)	6.0 (6.9)	4.0 (4.9)
Decode	256	47.4 (110.8)	49.4 (115.1)	47.4 (105.4)
Host-resident learned embeddings				
Prefill	1	4.4 (6.1)	4.4 (4.8)	—
Prefill	8	14.4 (20.1)	14.4 (19.5)	—
Decode	1	4.0 (4.9)	4.0 (4.7)	—
Decode	256	47.4 (110.8)	47.4 (115.1)	—

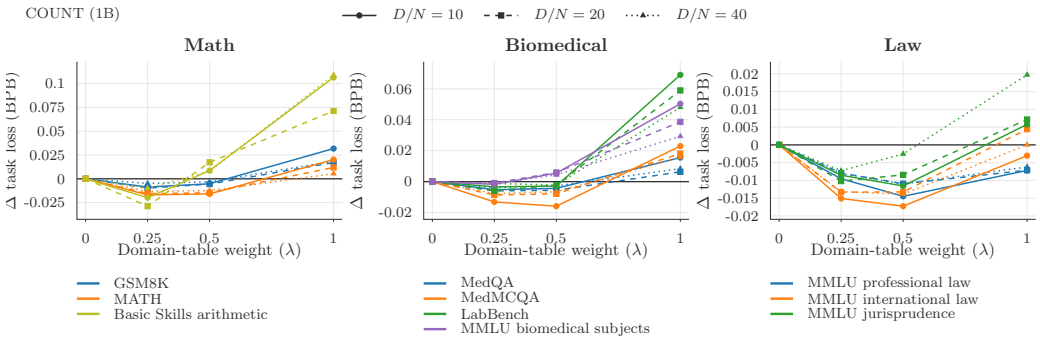

Figure 9: Full domain-table interpolation grid for 1B COUNT at $D/N = 10, 20, 40$. Each curve is relative to its own checkpoint's original-table score; negative values indicate improvement. MATH, LabBench, and MMLU biomedical subjects receive equal weight within their respective groups; the three law subjects remain separate.

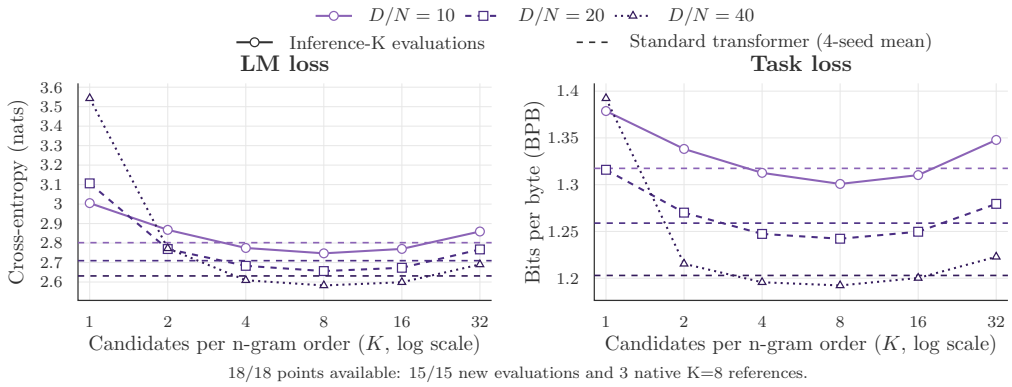

Figure 10: Inference-K sweep for the 1B COUNT model with its original 797.487 GB table fixed. LM loss averages token cross-entropy across 11 sources; task loss averages bits per byte across 74 task configurations. The logarithmic axis spans $K \in \{1, 2, 4, 8, 16, 32\}$ at all three budgets, with 15 new evaluations and three original $K = 8$ references. Lower loss is better.

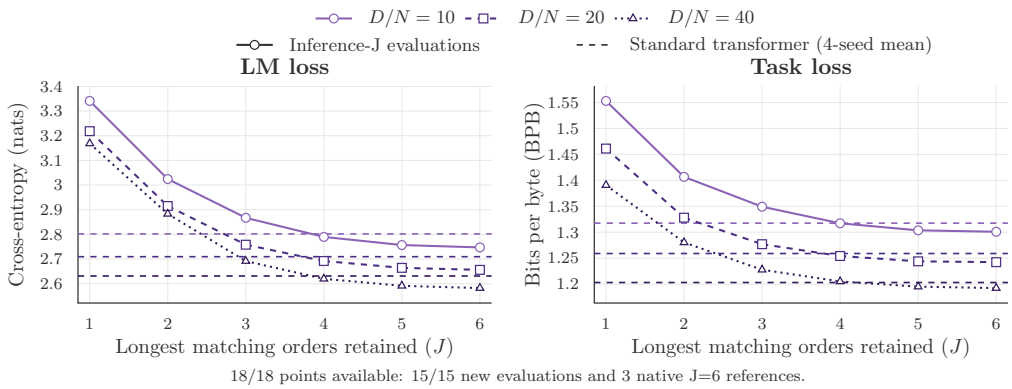

Figure 11: Inference-J sweep for the 1B COUNT model with its original 797.487 GB table and $K = 8$ fixed. Each context retains its J longest matching orders and divides their vector sum by J, with zeros for missing matches. LM loss averages token cross-entropy across 11 sources; task loss averages bits per byte across 74 task configurations. All 15 new evaluations and three original $J = 6$ references are included. Lower loss is better.



Figure 12: 190M COUNT trained with $n_{\max} = 3$ versus $n_{\max} = 6$ (same 17.1 GB table, $K = 8$, embedding amplification), at the post-decay endpoints $D/N \in \{10, 20, 40\}$. One run per setting.

B.5 Ablating embedding amplification

COUNT applies embedding amplification, $\sqrt{d}\,\mathrm{LayerNorm}(\cdot)$, to its combined input. Figure 13 ablates it within 190M COUNT: amplification lowers both LM and task loss at every budget. Figure 14 adds amplification to the standard transformer itself across the ladder: it barely moves the standard transformer, with some improvement at the smallest scales and some degradation at the largest. We therefore compare against the standard transformer without amplification, which gives the baseline its best performance at the largest scale and is the fairest comparison.

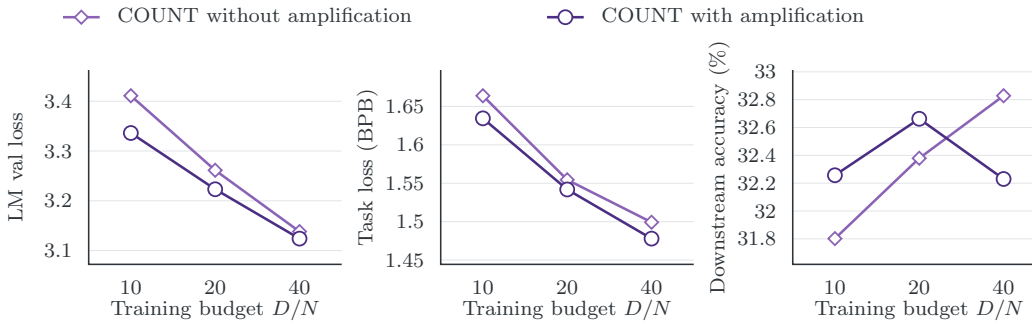

Figure 13: 190M COUNT with and without embedding amplification (17.1 GB table, $K = 8$, orders one through six) at $D/N \in \{10, 20, 40\}$. One run per setting.

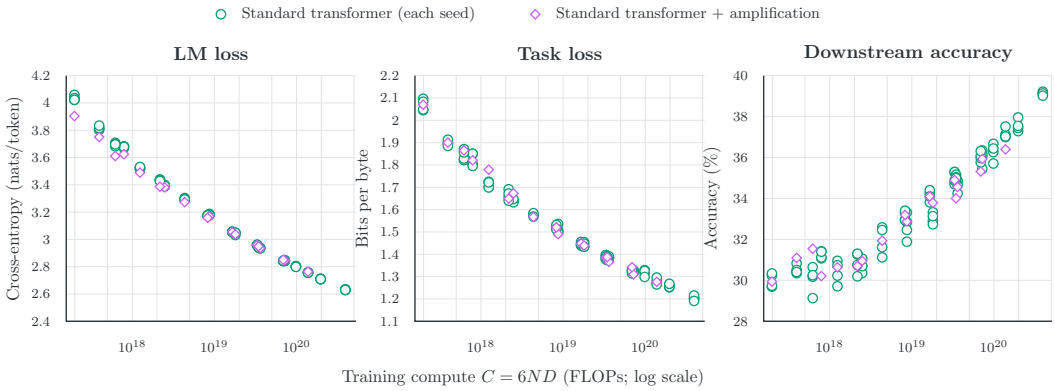

Figure 14: Embedding amplification across standard transformer scales, at nominal $D/N = 10, 20, 40$. Each standard-transformer seed is a separate point (84 endpoints across seven backbone sizes); the amplified arm has one run per size (21 endpoints). Metrics average sources or tasks, while each marker retains its individual training seed.