MEASURING PROGRESS IN DIFFUSION LANGUAGE MODEL PRETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

In contrast to autoregressive models, it is an open question how to meaningfully compare training runs of diffusion language models (DLMs). In this paper, we propose an empirical protocol for choosing a performance metric for early DLM web pretraining, so that training runs can be compared by the time they take to reach a certain target. We ask three questions of each possible metric: 1) Does the metric rank training recipes robustly under slight changes to the performance target? 2) Does the metric consistently improve during DLM pretraining, so that several options for setting a target are possible? And 3) Does the metric recover the known rankings of autoregressive models, whose text likelihood can be evaluated directly? We apply this protocol to the following metrics from the DLM literature: accuracy in several downstream tasks, generative perplexity and bits per byte, distributional similarity, decoded-text validity, repetition and diversity. We do so across seven popular masked and uniform DLM training recipes. In our experiments, downstream performance provides the clearest stable target for tracking progress. Generative perplexity and bits per byte also give comparatively stable rankings and consistently improve, but every tested autoregressive reference model incorrectly ranks at least one pair from the GPT-2 family (even after entropy matching). MAUVE also improves persistently and gives stable local rankings, but reverses two GPT-2 comparisons. Unigram and bigram JS preserve the entropy-matched AR orders tested here, although their local DLM rankings change more often. Validity and repetition mainly change early in training or favor random text, so they do not provide natural stopping targets. Alongside diversity measures, they remain useful checks on the DLM's samplers. Based on our findings, we release an open-source speedrun built around downstream HellaSwag accuracy, one of the metrics that meets our checks most consistently here.

Code and HellaSwag speedrun: anonymous.4open.science/r/speedrun-dlm-2F63

1 INTRODUCTION

Diffusion language models (DLMs) offer a promising alternative to the autoregressive language models that currently dominate the commercial landscape. They can revise several tokens in one model call and may therefore need fewer calls at generation time than autoregressive models, which generate text from beginning to end, one token per call. This advantage has motivated work resulting in models such as Gemini Diffusion, DiffusionGemma, LLaDA, Mercury, Dream, Seed Diffusion, TiDAR, and LLaDA2.0 (Google DeepMind, 2025; 2026; Nie et al., 2025; Inception Labs et al., 2025; Ye et al., 2025b; Song et al., 2025; Liu et al., 2026; Bie et al., 2025).

As of now, there is, however, no clear consensus on the best way to train and sample from DLMs. The corruption noise can, for instance, be uniform or masking, which affects whether a model can revise tokens produced at earlier denoising steps. Prominent training objectives include D3PM (Austin et al., 2021), variants in continuous-time for masked diffusion losses (Sahoo et al., 2024; Shi et al., 2024), score entropy (Lou et al., 2024), and diffusion duality variants (Sahoo et al., 2025; Deschenaux et al., 2026). In practice, large DLMs often use their own "homegrown" variants of these ideas (Nie et al., 2025; Ye et al., 2025b; Song et al., 2025; Liu et al., 2026; Bie et al., 2025).

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.

Evaluation of DLMs. While there is a great interest in comparing these various DLM recipes, evaluation of DLMs is fundamentally more challenging than in the autoregressive case. This is because their sequence-level log-likelihood (and thus their cross-entropy loss) involves a summation over all possible denoising paths and is therefore expensive to estimate. On the other hand, the loss functions of different DLM recipes, often derived from an evidence lower bound (ELBO) on said log-likelihood, are not directly comparable due to differences in the parameterization, the derivation, and the noise schedule. Large-scale releases typically report combinations of downstream-task performance, likelihood bounds and quality estimates of the generated text (Nie et al., 2025; Ye et al., 2025b; Song et al., 2025; Bie et al., 2025).

These different evaluation metrics come with different tradeoffs when used to compare heterogeneous DLM training recipes. Quantities based on likelihood bounds are often tied to a particular diffusion parameterization, noise process, and variational objective, so their numerical values need not be meaningfully comparable across model families (e.g., Sahoo et al. (2026)). Generative perplexity avoids this recipe dependence but introduces a new set of confounders: its conclusions can change with the autoregressive reference model and with the diversity (or entropy) of generated samples (Acquaviva, 2026; Pynadath et al., 2026; Franca & Tong, 2026). Downstream task performance can be measured with or without additional fine-tuning, and aims to anticipate the usefulness of the model more directly on concrete instances (e.g., Nie et al. (2025)). Distributional distances such as MAUVE and diversity-based metrics instead evaluate generated DLM samples by measuring proxies of their closeness to held-out text or other statistics (Pillutla et al., 2021; Shaib et al., 2025), but also require many design choices and can suffer from large variance. We defer an in-depth discussion of related work to Appendix A.

Pretraining speedruns. In this work, we ask: Among established evaluation metrics, which ones are most suitable for *tracking pretraining progress* across DLM recipes and their many design choices? Our goal is not to identify the metric that best showcases the capabilities of a fully trained or even post-trained model. Rather, we seek metrics that can provide a *consistent signal of progress early in training* and yield a *stable ranking of competing methods* as training proceeds.

This distinction matters because much of progress in machine learning comes from rapid iteration: researchers typically test many architectural choices, objectives, and ablations using relatively *small-scale* training runs, and only scale up the most promising candidates. In this setting, the relevant question is not simply how to evaluate the final model, but how one can reliably distinguish promising ideas *before* committing to an expensive full-scale training run. The nanoGPT speedrun benchmark of Jordan et al. (2024) provides a particularly clear example of this paradigm for *autoregressive* language models, where evaluation is much easier. There, a target threshold for the cross-entropy test loss is set and the training recipe requiring the least time to reach it wins. To date, there is neither a clear consensus nor an extensive study on what candidate metrics could be used to facilitate similar progress for DLM development.

Motivated by this, we investigate which existing metrics could offer reliable finish lines for an analogous *DLM speedrun*: If only one or a small number of metrics could be used to track early pretraining progress and rank competing recipes, which should they be? To this end, we evaluate candidate metrics against three basic criteria that, as we argue, any informative speedrun metric should satisfy. The execution of our screening protocol based on these three criteria marks the first extensive study comparing a wide range of established DLM evaluation metrics for measuring pretraining progress.

2 Protocol Setup and Main Results

Metric-screening protocol. Our main contribution is an empirical protocol for deciding which metrics can track training progress for DLMs.

1. **Stability: Is the recipe ranking robust to small changes in the target?**
   Every speedrun necessarily has to fix a numerical performance target at which the run is stopped. Our first desideratum asserts that for a given metric, the resulting ranking is robust and not mainly shaped by the exact numerical target used. Indeed, a small adjustment of the finish line should ideally only slightly influence the ranking of training recipes. Within the fixed pretraining experiment described below, we consider a range of carefully chosen numerical targets and their

perturbations based on the specific metric. We then measure how often such changes to the finish line also change the ranking of the seven DLM recipes. Methodology and results are detailed in Section 4.1.

2. **Monotonicity: Does the metric consistently reflect progress throughout training?** Within the same pretraining setup, we evaluate every metric at ten different training token budgets for each of the seven reference DLM recipes. For each recipe, we average the metric values across five training runs and measure the fraction of consecutive checkpoint pairs for which the metric improves. The spirit of this check is that a metric intended to track pretraining progress should generally assign better scores to later checkpoints on established diffusion training recipes. The results are in Section 4.2.
3. **Recovery: Does the metric make a ranking error on autoregressive model families?** Autoregressive models allow for direct access to their sequence likelihood. We order the GPT-2 (Radford et al., 2019), Pythia (Biderman et al., 2023), Llama 2 (Touvron et al., 2023) and Llama 3.1 (Grattafiori et al., 2024) families by their own likelihood on the same held-out validation FineWeb subset. A metric meant to compare pretrained models should not lead to a different ranking within any of these families. We report the results in Section 4.3.

In Section 3, we first recap the metrics to which we apply this protocol: They cover metrics based on downstream tasks (Section 3.1), autoregressive reference scorers (Section 3.2), statistics of generated text (Section 3.3), and distributional similarity (Section 3.4).

For the stability and monotonicity checks, we use seven established DLM recipes as reference points: SUBS (Sahoo et al., 2024) (masked), SEDD (Lou et al., 2024) and D3PM (Austin et al., 2021) with masking or uniform noise, and DUO (uniform) with its original curriculum or its top-k approximation (Sahoo et al., 2025; Deschenaux et al., 2026). We aggregate results across the seven recipes since each metric should ideally be screened for a diverse set of training runs. We also monitor and check non-aggregated results for the sake of illustration and completeness, but mostly defer them to the appendix throughout. The choice of DLM recipes inevitably shapes this assessment. Calibrating on a given set may favor metrics that recognize similar forms of progress in future recipes, while leaving them less sensitive to other improvements. However, we deliberately choose these seven recipes because they simultaneously represent a rather broad range of methodologies and the evolution of well-established recent DLM progress.

Main findings. Figure 1 summarizes the audit. Several downstream tasks, including HellaSwag, ARC-Easy, PIQA and LAMBADA, provide useful signals of pretraining progress. Finetuned MNLI also improves persistently, but its rankings change more often and we have not checked its AR ordering. Gen. PPL and BPB often improve and give reasonably stable rankings, yet every tested reference scorer reverses a GPT-2 comparison, even after entropy matching. Diversity and decoded-text validity checks complement these measures, although several plateau early or favor random text. MAUVE improves on 83% of adjacent checkpoint comparisons and gives few local ranking changes, but reverses two GPT-2 comparisons. Unigram and bigram JS recover the tested entropy-matched AR orders, with less stable local DLM rankings (Sections 3.4 and 4.3). The correlations between metric rankings give a complementary view in Figure 14.

Speedrun benchmark release. Building on this metric audit, we release an open-source DLM pretraining speedrun with a target of 27.5% HellaSwag `norm` accuracy (Section 5). It compares elapsed training time under the shared model, data and hardware setup below. Ten new seeds per recipe establish the baseline ranking and its sensitivity to nearby targets.

Experimental setting. For both the screening protocol and the speedrun release, we follow the scale of the well-established nanoGPT speedrun (Jordan et al., 2024), which has served as an accelerator of autoregressive pretraining research (for instance leading to the Muon optimizer, to name one example). We keep nanoGPT-2's backbone (12 layers, 12 attention heads, width 768) and make the minimal adjustments to turn it into a standard Diffusion Transformer (cf. Lou et al. (2024)). In particular, we switch to non-causal attention, add a noise-level embedding layer and adaptive normalization layers, and untie the final unembedding layer. This leads to a total parameter count of 169.6M. Training uses sequences of 1,024 tokens. Each DLM recipe is trained five times with seeds 1001–1005 on the same 3.3B GPT-2-token prefix of FineWeb (Penedo et al., 2024).

Metric / family	Local changes ↓	Comparison coverage ↑	Improving pairs ↑	AR order	Main limitation or tradeoff
ARC-Easy no-norm	1.8%	62%	79%	no inversion	Less coverage than HellaSwag norm
HellaSwag norm	4.5%	75%	76%	no inversion	Costlier than ARC-Easy and PIQA
HellaSwag no-norm	3.8%	78%	75%	no inversion	Slightly less progress than norm
PIQA norm / no-norm	1.7–2.2%	59–66%	67–70%	no inversion	Less progress than HellaSwag / ARC-Easy
LAMBADA	0.7%	54%	67%	no inversion	SEDD uniform near zero; coverage falls
MNLI, finetuned	15.9%	65%	81%	not measured	More local changes; finetuning needed
ARC-Easy norm	2.2%	66%	81%	at least one inversion	AR reversal despite local stability
BLiMP / MNLI direct	5.2–10.4%	74–75%	33–51%	at least one inversion	Limited progress; MNLI near chance
Structured, finetuned	1.3–14.5%	51–84%	0–52%	not measured	Little progress; two tasks stay at zero
Gen. PPL, six scorers	2.3–3.6%	65–70%	76–79%	at least one inversion	GPT-2 order reverses; cause unresolved
BPB, six scorers	5.3–5.9%	73–74%	71–73%	at least one inversion	GPT-2 order reverses; cause unresolved
MAUVE	1.7%	65%	83%	GPT-2 reversal*	Two GPT-2 pair orders reverse
Unigram / bigram JS	8.1–14.9%	68–78%	70–73%	Matched pairs agree*	More local changes; limited matched AR coverage
Semantic Vendi	4.4%	80%	81%	not an AR-order check	Variety alone does not establish quality
Within-sample repetition	1.9–3.5%	85–100%	33–41%	not an AR-order check	Random text already scores favorably
Other set diversity	1.5–4.0%	85–98%	0–51%	not an AR-order check	Little progress; duplicates stay at zero
Clean decoded text	11.1%	68%	68%	not an AR-order check	Early gains, then small changes

More favorable within a column Less favorable within a column

Column colors are relative, not rejection cutoffs. Ranges span metrics; their midpoints set the color.
* Distribution AR checks meet the entropy tolerance for GPT-2 and the Pythia 70M/160M pair.

Figure 1: **How the metric families behave in this experiment.** Columns follow our three checks: stability, monotonicity and recovery. Within each numerical column, green marks more favorable results and orange less favorable ones. Ranges span metrics within a row and are colored by their midpoint. Changes and coverage are averages over the 1–27-minute sweep. “Structured, finetuned” here contains all synthetic tasks. Full results are in Appendix E.

For the screening protocol, we choose 3.3B tokens as a practical early-pretraining budget. Our implementation reproduces each of the original DLM recipes listed above. For each recipe, we initialize the search from the hyperparameters of its official implementation and tune among close-by values. All recipes use AdamW with a constant learning rate after warmup. Batch size, warmup and noise schedule are recipe-specific (Appendix B.1). Complete runs take about 29–70 minutes on eight GH200 GPUs.

3 Metric Families

We assess four metric families against the three criteria of Section 2: downstream task accuracy, reference-model scores (Gen. PPL and BPB), diversity within and across generated samples, and similarity to held-out text. We introduce them here, before auditing them in Section 4. More details on these metrics are provided in Appendix C.

3.1 Downstream task accuracy

Since we train on the first 3.3B tokens of FineWeb, we look for abilities that can reasonably emerge at that stage: grammatical preferences, short-context completion, common-sense continuation and

simple transformations of synthetic sequences. Without finetuning, we evaluate the models on LAMBADA (final-word prediction) (Paperno et al., 2016), HellaSwag (commonsense continuation) (Zellers et al., 2019), ARC-Easy and ARC-Challenge (grade-school science) (Clark et al., 2018), PIQA (physical commonsense) (Bisk et al., 2020), BLiMP (grammatical acceptability) (Warstadt et al., 2020), and MNLI (relations between sentence pairs) (Williams et al., 2018). We refer to these as *direct* downstream tasks. In contrast, we also investigate several downstream tasks *after adaptation*: MNLI after finetuning, and we finetune (jointly, to save computation) each checkpoint on seven *synthetic* tasks, generated programmatically, each with a reference answer checked by exact match: addition, associative recall, bracket completion, copying, reversing, sorting and Sudoku.

Final accuracies on ARC-Challenge remain below chance, so we do not re-evaluate it over the full trajectory. For all other trajectories we evaluate at the same ten token-count checkpoints, for all seven DLM recipes and across five independent training runs. Appendices C.1.2 and C.1.3 describe the evaluation without and with finetuning, respectively.

For HellaSwag, ARC-Easy and PIQA, we use the prompts and two scoring conventions of the Language Model Evaluation Harness (LM Eval Harness): `norm` divides an answer's score by its character count, while `no-norm` does not apply such a normalization (Biderman et al., 2024). We adapt these conventions to DLMs by keeping the prompt visible, corrupting each answer, and asking the model to recover it (Nie et al., 2025; Sahoo et al., 2024). If c is the prompt and a is one answer, its score is

$$\widehat{s}(a) = \frac{1}{K d(a)} \sum_{k=1}^{K} L_\theta^{\text{recipe}}(a; c, \omega_k(a), t_k), \qquad \widehat{a} = \arg\min_a \widehat{s}(a), \tag{1}$$

where L_θ^{recipe} is the recipe's loss for recovering the clean answer a from the input $(c, \omega_k(a))$ at noise level t_k. Here, $\omega_k(a)$ is one randomly corrupted version of the answer. We set $d(a) = |a|_{\text{char}}$ for `norm` and $d(a) = 1$ for `no-norm`. For clarity, we provide a step-by-step illustration in Appendix C.1.2. This answer-only corruption convention is used, for instance, in (Nie et al., 2025; Sahoo et al., 2024). Another possible convention is to also corrupt the context (Deschenaux et al., 2026). In our experiments, the latter often led to close-to-chance accuracies, which is why we adopt the answer-only corruption.

3.2 Reference score

Gen. PPL and BPB are common alternatives to the unavailable DLM text likelihood. An autoregressive reference model R (the *scorer*) assigns a loss $L_R(x)$, namely the negative log-likelihood, to a sample x generated by the DLM. For a bank X of samples with $N_R(x)$ tokens (under the scorer's tokenizer) and $B(x)$ decoded UTF-8 bytes for sample x, they measure

$$\text{GenPPL}_R = \exp\left(\frac{\sum_{x\in X} L_R(x)}{\sum_{x\in X} N_R(x)}\right), \qquad \text{BPB}_R = \frac{\sum_{x\in X} L_R(x)}{\log(2) \sum_{x\in X} B(x)}. \tag{2}$$

In short, Gen. PPL divides by scorer tokens, while BPB divides by bytes to account for tokenizer-dependence. We compare six standard scorers, including GPT-2 Large and Qwen models of 1.7B and 7B parameters (see Appendix C.2).

It is well-known that these metrics can favor text that is less diverse. For instance, once a pattern repeats, an autoregressive scorer may find its next repetitions easier to predict. Following recommendations by Pynadath et al. (2026) and Acquaviva (2026), we therefore compare Gen. PPL and BPB of sample banks at the same token entropy. For a tokenized sample $x_i = (x_{i,1}, \ldots, x_{i,T_i})$, let $\widehat{p}_i(v) = T_i^{-1} \sum_{t=1}^{T_i} \mathbf{1}\{x_{i,t} = v\}$ be the frequency of token v. The bank entropy is $\overline{H}(X) = -n^{-1} \sum_i \sum_v \widehat{p}_i(v) \log \widehat{p}_i(v)$. We use $n = 64$ samples for DLM banks ($n = 128$ for AR banks), natural logarithms, and samples cut to at most 1,023 GPT-2 tokens (Appendix C.2). We tune the temperature of a DLM to sample a bank X such that $|\overline{H}(X) - 5.471| \leq 0.02$, where 5.471 is the median entropy of held-out OpenWebText (Pynadath et al., 2026) (see Figures 6 and 7).

3.3 Diversity within and across samples

Motivated by the aforementioned diversity-quality tradeoff, we audit metrics that measure how varied the generated text is beyond single-token entropy. We include variety *within* samples by using

repeated n-grams and compression measures. As for variety *across* samples, we use Self-BLEU (n-gram overlap) and semantic Vendi (feature diversity, where the features are obtained via embeddings based on `all-mpnet-base-v2`) (Zhu et al., 2018; Friedman & Dieng, 2023; Shaib et al., 2025). We additionally retain one simple decoding check: the fraction of samples without replacement symbols or unwanted control characters. This check asks whether the sampler returns readable text at all. All precise definitions are given in Appendix C.3.

3.4 Similarity to held-out text

Of course, variety alone does not reveal whether samples resemble natural text. MAUVE compares the distributions of generated and held-out samples through clustered representations from a pretrained model (Pillutla et al., 2021). Unigram and bigram Jensen–Shannon divergence instead compare how often words and pairs of words occur (see Appendix C.4 for both). We use 1,000 generated and 1,000 held-out samples for all three, with GPT-2 Large representations for MAUVE. Temperature is tuned toward natural-text token entropy 5.471; 289 of 350 DLM generated banks of 1,000 samples meet the ± 0.02 tolerance, and the remaining absolute gaps are 0.020–0.082 (Appendix C.4).

4 Results of metric auditing

We now audit the four metric families against the three criteria (stability, monotonicity, recovery) of Section 2. This section explains the results in Figure 1. All the detailed results are in Appendix E.

4.1 Stability: Robustness to small changes in the target

One of the main goals of this work is to audit standard evaluation metrics to see which could define a useful numerical target in a small-scale pretraining speedrun. Our first criterion therefore asks whether a metric produces *robust* rankings near possible numerical targets. For instance, if the target is changed from, say, 27% to 28% accuracy, are the rankings affected or not? While recipes may indeed learn at different paces, stable rankings are reassuring when testing ideas at small scale before scaling up: if even a small target adjustment flips the ranking, the conclusion about what works at this scale becomes more fragile. To define such a stability criterion, the main choices are: (i) Are we interested in rankings by training time or training tokens? (ii) How to choose the possible targets and their perturbations? (iii) Which changes in ranking can we compare, and when do they matter most?

Regarding (i), we are interested in a speedrun, so we rank recipes by training time to a target. Different recipes process the same number of tokens at different speeds: about 0.8–1.9 million tokens per second across eight GPUs in our setup (Table 1).

To address (ii), at each minute from 1 to 27, we assign each metric a target equal to the best recipe score at that time. All seven recipes have observations covering this range and its $\pm 5\%$ neighborhood. To obtain these per-minute scores, we linearly interpolate the five-run averages between the ten measured checkpoints (Appendix B.3). The target's perturbation then reflects both variation across the five training seeds and changes in the best score when the budget moves by $\pm 5\%$ (Appendix D.1).

As for (iii), we compare a pair of recipes only when its relative ranking is known at both targets. This ranking is known if both recipes reach the target, or if one reaches it while the other has been observed at least as long without passing. If neither reaches it, or the non-finisher's run ends too soon, longer training could still reveal either ranking. We call the fraction of the $\binom{7}{2} = 21$ recipe pairs we can compare *coverage*. Among the comparable pairs, we count the percentage whose order changes. Finally, we believe that it is reasonable to take into account that a summary statistic of these stability percentages should prioritize *late* ranking stability over *early* stability, because the behavior close to initialization is less relevant to scaling up training. We therefore report the full stability curves against both coverage and training time (Figure 2).

Overall, **this first check is reassuring for several metrics**: many average fewer than 5% pairwise ranking changes, although some reach about 10–15%. The curves help distinguish them further: few changes across many comparable pairs, persisting as the budget grows, are more useful than few changes based on only a handful of comparisons. HellaSwag, for example, has at most 0.1% changes while covering 71–83% of pairs between 10 and 20 minutes. ARC-Easy and PIQA also have regions

with few changes and broad coverage. LAMBADA has similarly few changes but lower coverage: 29% at 15 minutes, against 71% for HellaSwag.

The less favorable results include finetuned MNLI (about 16% changes at 65% coverage), direct MNLI (10% at 75%), and unigram and bigram JS (8–15% at 68–78%), rendering them less suitable. MAUVE combines 1.7% changes with 65% coverage. LAMBADA presents a different tradeoff: almost no changes, but only 54% coverage on average. These differences already help compare metrics, without clearly ruling out most of them.

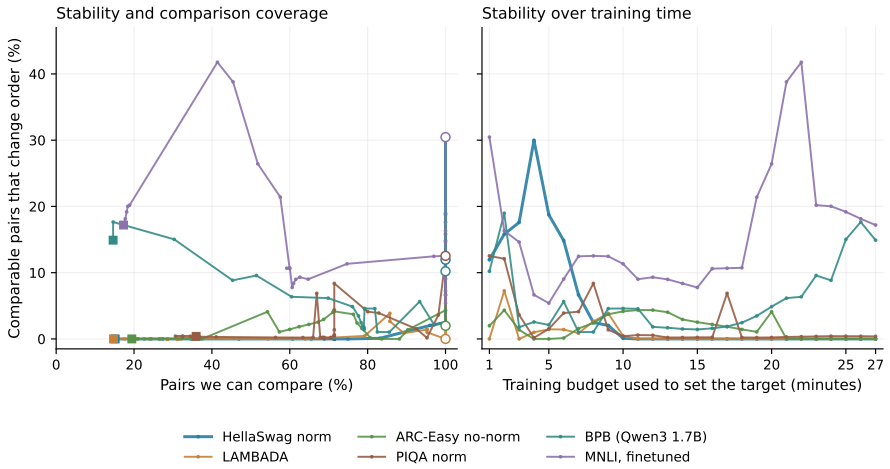


Figure 2: **The same selected metrics viewed against coverage and training time.** Each dot tests targets near the best score at one budget, from 1 to 27 minutes. Left: the lower-right region combines broad coverage with few order changes; open circles mark 1 minute and squares 27 minutes. Right: the same change rates show when stability persists. See Figure 8 to visualize all metrics.

4.2 Monotonicity: Reflecting progress throughout training

We next ask whether a metric consistently reflects progress as pretraining proceeds, as in question 2) of our protocol. This helps us find metrics that offer natural targets at different speedrun training budgets. If a metric plateaus early or takes too long to improve, it gives us little room to define a useful speedrun there. For each of the seven reference DLM recipes, we evaluate the metric at the same ten checkpoints (determined by training token budgets), average its value over five independent training runs, and count the fraction of the adjacent checkpoint pairs for which the mean score improves in the preferred direction. We then average this fraction across recipes, so that 100% means that every recipe improves at every consecutive checkpoint. The exact setup is described in Appendix D.2.

While the stability check alone gave no clear reason to rule out a metric, this second check separates them more clearly.

Several repetition and diversity measures offer poor progress targets. Random text already contains many distinct words and word groups. These measures can therefore start at favorable values and then *get worse* as the text becomes more like natural language. They remain useful safeguards against a model repeating the same text again and again, but we observe no such collapse in the runs here. The clean-decoding check also offers little room for a target after its initial rise. Semantic Vendi is an important exception: it improves on 81% of adjacent checkpoint pairs, although variety in meaning alone does not establish text quality (Table 8 and Figure 11).

The synthetic finetuned tasks and direct MNLI also offer poor progress targets. The accuracies on synthetic tasks improve on only 0–52% of adjacent checkpoint pairs, and direct MNLI accuracy on 33%. BLiMP is less consistent too, at 51%. Their individual curves show that these tasks are too easy, too hard, or fluctuate a lot (Figure 9). This leaves less room for a useful finish line.

Several other downstream tasks remain promising. The accuracy on HellaSwag `norm` improves on 76% of pairs, on ARC-Easy `no-norm` 79%, and on PIQA 67–70% (Table 8). LAMBADA accuracy improves 67% of the time but one recipe stays near zero accuracy (Figure 9), which was

already impacting its coverage in the stability check. Finetuned MNLI accuracy also improves persistently (81%), but Section 4.1 shows that it has less stable rankings.

Reference scores and distributional similarity also keep improving. Gen. PPL improves on 76–79% of pairs, BPB on 71–73%, MAUVE on 83%, and unigram and bigram JS on 70% and 73%, respectively (Table 8 and Figures 10 and 12). These metrics therefore remain of interest after the first two checks. The final check asks whether they also recover a known model ordering.

4.3 Recovery: Preserving within-family autoregressive likelihood rankings

Several metrics remain promising after the first two checks (HellaSwag, PIQA, ARC-Easy, Gen. PPL, BPB, MAUVE and Semantic Vendi). We finish our screening by asking which metrics recover the rankings of popular autoregressive (AR) models, whose text likelihood is directly accessible. This provides an external consistency check: a metric used to measure DLM pretraining progress should ideally agree with this reference ordering on established AR families.

We consider four AR families: GPT-2 from Small to XL, Pythia from 70M to 2.8B, Llama 2 from 7B to 70B, and Llama 3.1 from 8B to 70B. We establish the within-family order from each model's likelihood on the same held-out FineWeb text. For these models, it agrees with the expected ordering by size. Measuring it also tells us which pairs are closer, and may therefore be harder to distinguish: the relative NLL gaps range from about 3% to 30%. Appendix D.3 and Table 7 give the procedure and every gap, and Appendix E.3 relates these gaps to the observed reversals.

HellaSwag and PIQA (both `norm` and `no-norm`), ARC-Easy `no-norm`, and LAMBADA recover every tested order. However, one recipe stays near zero on LAMBADA, contributing to its lower improvement and coverage scores in the previous parts of the audit. We have not applied the AR check to finetuned tasks, as it would require an additional optimization run for every AR model. Nor do we apply it to diversity metrics, as these metrics are not meant to recover rankings of AR models by likelihood: diversity tests a different property (Appendix D.3).

Perhaps surprisingly, **metrics based on autoregressive scorer models (Gen. PPL and BPB) do *not* recover the GPT-2 ranking** in our experiments. Every tested reference scorer reverses at least one pair under each metric, even after matching token entropy within the prescribed tolerance (Table 10). When we repeatedly resample the generated-sample banks, the reported reversals recur in 59.2–95.9% of evaluations for Gen. PPL and 80.5–99.7% for BPB, depending on the scorer.

Based on our reading of the experiments, the most plausible explanation is a combination of the following. i) Gen. PPL and BPB being so sensitive to entropy that temperature tuning would be required to match it to unreasonable precision (especially for frequent evaluation in a speedrun). ii) Token-entropy being a poor proxy for sequence-level entropy, which should be matched in an ideal scenario but is intractable (Pynadath et al., 2026). In the GPT-2 comparison, where all four banks of 1,000 samples meet the entropy tolerance, MAUVE also reverses two adjacent pairs, whereas unigram and bigram JS recover all three orders (Appendix E.3 and Table 12). All three metrics also recover the matched-entropy Pythia 70M/160M order. The remaining AR comparisons exceed the tolerance for at least one model, so they cannot separate metric behavior from entropy mismatch.

5 From the audit to a HellaSwag speedrun

As our audit revealed, the downstream accuracies on HellaSwag, PIQA (both `norm` and `no-norm`), and ARC-Easy `no-norm` offer the most suitable targets. LAMBADA also keeps improving with few ranking changes, but covers fewer recipe comparisons and leaves one recipe near zero.

To put the audit into practice, we release a **HellaSwag speedrun with a target of 27.5% accuracy** at `anonymous.4open.science/r/speedrun-dlm-2F63`. Among the metrics retained by the audit, we choose HellaSwag `norm` for its widespread use in the DLM literature. The target is 2.5 percentage points above chance, where useful differences already emerge between recipes. We evaluate seven recipes with ten new training seeds each, separate from the five used in the audit. Figure 3 shows their ranking and its sensitivity to nearby targets, providing reference results for future improvements in DLM pretraining. See Appendix F for the protocol and training curves.

We highlight some conclusions of the speedrun benchmark to illustrate the usefulness of our audit. First, the ranking is consistent with prior evidence, e.g., Sahoo et al. (2024) report lower perplexity

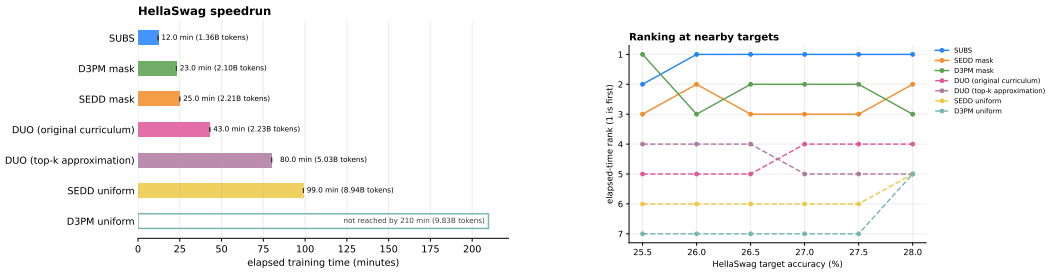


Figure 3: **A HellaSwag speedrun instance.** Time when ten-seed average accuracy reaches 27.5% (left; bars: $\pm$ one standard deviation of training times at the selected checkpoints) and ranks at nearby targets (right; solid: mask noise, dashed: uniform noise). Timing details are in Appendix F.2; code: `anonymous.4open.science/r/speedrun-dlm-2F63`.

bounds for SUBS than SEDD mask. Our results extend these comparisons to early training time to a common downstream target (Figure 3). SUBS reaches 27.5% in 12 minutes (vs. 23 minutes for D3PM mask, 43 minutes for DUO original). The masked recipes retain the first three positions across targets from 26.5% to 28%. Their advantage also appears across all five retained audit metrics: HellaSwag and PIQA (both `norm` and `no-norm`), and ARC-Easy `no-norm` (Figure 9).

Second, token budget and training time to a given performance can lead to very different conclusions. For instance, SEDD mask and uniform process both have similar throughput (1.5M tokens per second), but uniform needs about four times as many tokens to reach 27.5%, taking 99 minutes rather than 25. Thus, implementation improvements could obviously change the ranking in favor of uniform DLMs. Another example is DUO top-k, which is 21% more token-efficient than DUO original, but ends up, in this specific setup, needing more than twice as many tokens to reach 27.5% accuracy. These results are, of course, conditioned on our tuning budget, re-implementations of the original methods, and to the specific data and setup considered here. They should thus not be misunderstood as a final word on which method is absolutely better than another. In any case, these are the kinds of trade-offs that a shared speedrun can make visible. We hope it gives the community a place to collaboratively compete on a common target, share what works and build on one another's improvements.

6 Limitations and Outlook

Principled evaluation of DLMs remains an open problem in the literature. There is no single perfect choice: any set of metrics favors some properties over others. We aim to help the community understand these tradeoffs through three checks: 1) stability of DLM rankings under slight adjustments of the target, 2) continued improvement during DLM pretraining, and 3) agreement with known rankings of autoregressive models where applicable. We apply these checks to a nanoGPT-inspired web-pretraining setup.

There are several natural (and, we believe, important) ways to extend our study. Once a metric is chosen, the numerical target could be set following transparent principles, for example by favoring ranges where nearby targets preserve the recipe order, as illustrated on the right of Figure 3, by using the score achieved by a reference model such as GPT-2 Small, or by favoring targets where progress is steep and consistent. Sampler-dependent evaluation also deserves a broader separate study. For the corresponding metrics investigated here, we pair each DLM recipe with the sampler used in its reference implementation and fix the exact configuration in the released code; other samplers could lead to different results. Another aspect to be considered in a large study is the variance when scoring answers to multiple-choice questions. In this work, we use the same number of random corruptions for every DLM recipe, yet equal evaluation budgets may lead to different variances across different losses and corruption processes. Appendix F.4 reports the variation observed here, and one could calibrate these estimators recipe by recipe. Finally, audits at larger scales, more extensive tuning, combinations of metrics, and downstream tasks designed for early web pretraining could each extend these findings. We hope the released speedrun offers a place to test these ideas and identify promising new DLM recipes as the benchmark evolves.

AI DISCLOSURE

In this work, we used generative AI tools to provide feedback on the research methodology—for example, to discuss choices regarding the scale of the benchmark, possible implementations of parts of the repository, or whether alternative interpretations of the results should be considered. We also used these tools to help design experiments by providing them with the plans for what we wanted to build, and to organize the resulting outputs so that they were ready for diagnosis and interpretation by us.

We did not use generative AI tools to develop the conceptual framework of the work, including the three checks used in the audit, or to make the final interpretation of the results. The remaining required disclosure categories are not applicable to this work.

Additionally, we used generative AI tools to create or modify scientific figures or images, create or edit software code, review the existing literature to check whether we had missed relevant work, brainstorm, search for information, and edit the research paper to improve readability.

We reviewed all AI-assisted work. Generated code was tested and reviewed against official reference implementations; suggestions for improving the readability of the paper were reviewed and incorporated individually only when we agreed with them; suggested literature that we might have missed was read and checked for relevance; and the post-processing of experimental results was also checked by us. The research methodology, conceptual framework, and final interpretation of the results remain our own.

We take responsibility for the final content of this work, including any text, claims, code, figures, or other artifacts produced with the aid of generative AI.

REFERENCES

Sam Acquaviva. Fixing flow model evals. https://samacquaviva.com/projects/flow-evals/, 2026. Accessed: 2026-08-19.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International Conference on Learning Representations*, 2024.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, 2021.

Thomas Bertolani, Davide Bucciarelli, Leonardo Zini, Marcella Cornia, and Lorenzo Baraldi. Diffusion language models: An experimental analysis. *arXiv preprint arXiv:2606.19475*, 2026.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, 2023.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.0: Scaling up diffusion language models to 100B. *arXiv preprint arXiv:2512.15745*, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large scale autoregressive language modeling with Mesh-Tensorflow. Zenodo, https://doi.org/10.5281/zenodo.5297715, 2021.

Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language GANs falling short. In *International Conference on Learning Representations*, 2020.

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. *arXiv preprint arXiv:1803.05457*, 2018.

George E. Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, Juhan Bae, Justin Gilmer, Abel L. Peirson, Bilal Khan, Rohan Anil, Mike Rabbat, Shankar Krishnan, Daniel Snider, Ehsan Amid, Kongtao Chen, Chris J. Maddison, Rakshith Vasudev, Michal Badura, Ankush Garg, and Peter Mattson. Benchmarking neural network training algorithms. *arXiv preprint arXiv:2306.07179*, 2023.

Justin Deschenaux, Caglar Gulcehre, and Subham Sekhar Sahoo. The diffusion duality, chapter II: Ψ-samplers and efficient curriculum. In *International Conference on Learning Representations*, 2026.

Antonio Franca and Alexander Tong. Hacking generative perplexity: Why unconditional text evaluation needs distributional metrics. In *ICML Workshop on Structured Probabilistic Inference & Generative Modeling*, 2026.

Dan Friedman and Adji Bousso Dieng. The Vendi score: A diversity evaluation metric for machine learning. *Transactions on Machine Learning Research*, 2023.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry Hungry Hippos: Towards language modeling with state space models. In *International Conference on Learning Representations*, 2023.

Google DeepMind. Gemini Diffusion. https://deepmind.google/models/gemini-diffusion/, 2025. Accessed: 2026-08-19.

Google DeepMind. DiffusionGemma. https://deepmind.google/models/gemma/diffusiongemma/, 2026. Accessed: 2026-08-19.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Etrit Haxholli, Yeti Z. Gurbuz, Oğul Can, and Eli Waxman. Efficient perplexity bound and ratio matching in discrete diffusion language models. In *International Conference on Learning Representations*, 2025.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Arseny Ivanov, Sergei Kholkin, Vladislav Gromadskii, Grigoriy Ksenofontov, Ivan Oseledets, and Alexander Korotin. TUBE: Tangent upper bound on evidence for discrete diffusion language models. *arXiv preprint arXiv:2605.24292*, 2026.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *International Conference on Machine Learning*, 2024.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, et al. modded-nanogpt: Speedrunning the NanoGPT baseline. https://github.com/KellerJordan/modded-nanogpt, 2024. Accessed: 2026-08-19.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, and Kangwook Lee. ParallelBench: Understanding the trade-offs of parallel decoding in diffusion LLMs. In *International Conference on Learning Representations*, 2026.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Ce Zhang, and Pavlo Molchanov. TiDAR: Think in diffusion, talk in autoregression. In *Conference on Machine Learning and Systems*, 2026.

Lang Liu, Krishna Pillutla, Sean Welleck, Sewoong Oh, Yejin Choi, and Zaid Harchaoui. Divergence frontiers for generative models: Sample complexity, quantization effects, and frontier integrals. In *Advances in Neural Information Processing Systems*, 2021.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024.

Peter Mattson, Christine Cheng, Gregory F. Diamos, Cody Coleman, Paulius Micikevicius, David A. Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim M. Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. MLPerf training benchmark. In *Conference on Machine Learning and Systems*, 2020.

Clara Meister and Ryan Cotterell. Language model evaluation beyond perplexity. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, 2025.

Theo X. Olausson, Metod Jazbec, Xi Wang, Armando Solar-Lezama, Christian A. Naesseth, Stephan Mandt, and Eric Nalisnick. A tale of two temperatures: Simple, efficient, and diverse sampling from diffusion language models. In *Conference on Language Modeling*, 2026.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Annual Meeting of the Association for Computational Linguistics*, 2016.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics*, 2002.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, 2024.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. MAUVE: Measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems*, 2021.

Krishna Pillutla, Lang Liu, John Thickstun, Sean Welleck, Swabha Swayamdipta, Rowan Zellers, Sewoong Oh, Yejin Choi, and Zaid Harchaoui. MAUVE scores for generative models: Theory and practice. *Journal of Machine Learning Research*, 24(356):1–92, 2023.

Tiago Pimentel, Clara Isabel Meister, and Ryan Cotterell. On the usefulness of embeddings, clusters and strings for text generation evaluation. In *International Conference on Learning Representations*, 2023.

Patrick Pynadath, Jiaxin Shi, and Ruqi Zhang. Generative frontiers: Why evaluation matters for diffusion language models. *arXiv preprint arXiv:2604.02718*, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin T Chiu, and Volodymyr Kuleshov. The diffusion duality. In *International Conference on Machine Learning*, 2025.

Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukić. Scaling beyond masked diffusion language models. In *International Conference on Machine Learning*, 2026.

Yash Shah, Abhijit Chakraborty, and Vivek Gupta. CaRE: Compute-aware remasking evaluation protocol for masked diffusion language models. *arXiv preprint arXiv:2607.24763*, 2026.

Chantal Shaib, Venkata S Govindarajan, Joe Barrow, Jiuding Sun, Alexa Siu, Byron C Wallace, and Ani Nenkova. Standardizing the measurement of text diversity: A tool and comparative analysis. In *International Joint Conference on Natural Language Processing (System Demonstrations)*, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, 2024.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed Diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Luhan Tang, Longxuan Yu, Shaorong Zhang, and Greg Ver Steeg. Is your diffusion sampler actually correct? A sampler-centric evaluation of discrete diffusion language models. In *International Conference on Machine Learning*, 2026.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Gilad Turok, Chris De Sa, and Volodymyr Kuleshov. DUEL: Exact likelihood for masked diffusion via deterministic unmasking. *arXiv preprint arXiv:2603.01367*, 2026.

Dimitri von Rütte, Janis Fluri, Omead Pooladzandi, Bernhard Schölkopf, Thomas Hofmann, and Antonio Orvieto. Scaling behavior of discrete diffusion language models. In *International Conference on Learning Representations*, 2026.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *International Conference on Learning Representations*, 2025a.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025b.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, 2019.

Hengxiang Zhang, Jiaxi Ren, Renchunzi Xie, and Hongxin Wei. Understanding evaluation illusion in diffusion large language models. *arXiv preprint arXiv:2606.29228*, 2026.

Linhao Zhong, Linyu Wu, Wen Wang, Yuling Xi, Chenchen Jing, Jiaheng Zhang, Hao Chen, and Chunhua Shen. Efficient self-evaluation for diffusion language models via sequence regeneration. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018.

APPENDIX CONTENTS

A Relation to earlier evaluation work **16**

B Reference DLM training recipes **18**

B.1 Training setup and hyperparameters 18

B.2 Departures from the official implementations 19

B.3 Saved checkpoints 19

B.4 Training and validation losses 19

C Metric families **20**

C.1 Downstream tasks 21

C.2 Reference scores: Gen. PPL and BPB 24

C.3 Diversity within and across samples 26

C.4 Similarity to held-out text 27

D Metric screening protocol **27**

D.1 Stability: robustness to small changes in the target 27

D.2 Monotonicity: improvement across shared checkpoints 28

D.3 Recovery: autoregressive calibration 29

E Audit results by criterion **30**

E.1 Stability: order changes and coverage of every metric 30

E.2 Monotonicity: improvement of every metric over training 32

E.3 Recovery: autoregressive orders reversed by each metric family 34

E.4 Redundancy across families 37

F HellaSwag speedrun: protocol and results **37**

F.1 Benchmark setup and target 37

F.2 Locating and verifying the finish time 38

F.3 Training trajectories and benchmark results 38

F.4 Variability across training seeds and evaluation corruptions 40

A Relation to earlier evaluation work

Evaluation of DLMs spans several related but distinct questions: whether a learned model assigns high probability to held-out text, whether its samples resemble natural text, whether a finite-step sampler faithfully represents the learned model, and whether the model performs well on downstream tasks. Unsurprisingly, these questions need not induce the same ranking of models. Then, given such a performance measure, one may ask how quickly a training recipe reaches a chosen level of performance. Our focus is on this last question. Rather than proposing yet another performance metric, we ask: Among established metrics, which ones are suitable for *tracking pretraining progress* across DLM recipes and their many design choices? Most of the evaluation work below scores fully trained models once, often on demanding downstream benchmarks. A speedrun instead needs a measure that already separates recipes early in pretraining, which is the property our audit tests.

Evaluation practice in diffusion language models. Early discrete diffusion language models commonly emphasized variational likelihood bounds. D3PM (Austin et al., 2021), simplified masked diffusion (Shi et al., 2024), MDLM (Sahoo et al., 2024), SEDD (Lou et al., 2024), and DUO (Sahoo et al., 2025) all report a likelihood- or perplexity-like quantity based on variational bounds (often already supplemented by Gen. PPL or generated-text metrics). Such quantities are natural when evaluating a fixed model family, but the corresponding objectives differ in parameterization, noise process, and their variational loss, which can render direct comparison across families uninformative. Thus, it has since become the standard to supplement reported losses with family-agnostic Gen. PPL or generated-text metrics.

More recent work increasingly adds downstream task evaluation to their results. This is the case both in academic comparisons of diffusion formulations (Sahoo et al., 2026; von Rütte et al., 2026) and in large-scale systems such as LLaDA (Nie et al., 2025), Mercury (Inception Labs et al., 2025), Dream (Ye et al., 2025b), Seed Diffusion (Song et al., 2025), TiDAR (Liu et al., 2026), and LLaDA2.0 (Bie et al., 2025). At this scale, task performance and generation speed are often central evaluation quantities. Recent experimental comparisons of already-trained DLMs likewise rely on suites of downstream tasks and controlled inference settings (Bertolani et al., 2026). Our experiments instead concern a much earlier stage of web pretraining at academic research scale. Here, many capabilities are still near chance and the purpose of evaluation is to measure *pretraining progress* rather than final model quality.

Likelihood evaluation and comparisons across DLM families. One direction tries to overcome the comparability issue of likelihood bounds by tightening those bounds. Haxholli et al. (2025) derive tighter perplexity bounds for discrete diffusion, while Ivanov et al. (2026) introduce a complementary *upper* bound on text-likelihood for several discrete latent-variable models. For a certain class of masked diffusion samplers, exact likelihood evaluation can also become possible: Turok et al. (2026) derive exact sequence likelihood for a class of deterministic unmasking schemes. Complementarily, Zhong et al. (2026) propose DiSE, which uses token-regeneration probabilities conditioned on the full sequence as an efficient proxy for conditional likelihood.

These advances improve current likelihood bounds, but by themselves do not remove the underlying comparability problem in general *across* diffusion formulations, such as masked and uniform ones. As an illustration, Sahoo et al. (2026) show that perplexity (estimated based on likelihood bounds) is informative within a diffusion family but can be misleading across families: their uniform-state model achieves worse validation perplexity than masked diffusion while performing better on GSM8K after supervised finetuning, and different families occupy different parts of the speed–quality frontier. Similarly, scaling studies find that the relation between model size, data, compute, and loss heavily depends on the underlying diffusion formulation (von Rütte et al., 2026). These observations motivate evaluating candidate progress measures in a way that is less biased by the recipe's native loss.

Generative perplexity, diversity, and distributional evaluation. Gen. PPL avoids reliance on family-specific training losses by scoring generated samples with an external reference autoregressive model. For such a metric, it is important to take into account the dependency on the diversity of the model. Caccia et al. (2020) advocate comparing quality–diversity curves obtained by varying sampling temperature, rather than evaluating a model at a fixed temperature (and hence diversity). This has been shown to be particularly important for DLMs: Acquaviva (2026) shows that relatively

small changes in token entropy can produce large changes in Gen. PPL, and that the choice of reference model can alter model comparisons. Pynadath et al. (2026) characterize this tradeoff using Gen. PPL–entropy frontiers, motivated by the decomposition of KL divergence into cross-entropy and entropy terms.

An obvious fix, entropy matching, removes one important confounder but still does not turn Gen. PPL into a direct distributional or semantic-quality metric. Franca & Tong (2026) construct deliberately simple "dummy" generators that obtain competitive Gen. PPL at non-degenerate empirical entropy despite producing visibly poor text. They consequently advocate direct distributional comparisons between generated and reference (held-out) text samples. MAUVE (Pillutla et al., 2021) is one prominent example for such a distributional metric. It embeds generated and reference samples and compares their induced distributions. Other work has discussed lexical, different distributional, and representation-based statistics for generated language (Meister & Cotterell, 2021; Pimentel et al., 2023; Shaib et al., 2025). These metrics characterize different aspects of generated text, but their usefulness for sample evaluation does not by itself establish their suitability as speedrun finish lines. This requires assessing whether a calibrated target marks meaningful progress and yields reliable comparisons across training recipes. We therefore include several such measures in our audit, while using decoding validity and repetition primarily as safeguards against degenerate sampling behavior.

Sampler dependence in training evaluation. Evaluating training progress through generated text is complicated by the fact that these measurements assess a trained model together with a sampling procedure. The contribution of the sampler can be substantial: Tang et al. (2026) show that few-step sampling can introduce distributional errors even with an exact denoiser; PARALLELBENCH (Kang et al., 2026) constructs tasks with strong inter-token dependencies specifically to expose the accuracy degradation induced by aggressive parallel decoding; and Olausson et al. (2026) show that tempering confidence-based position selection can improve sample diversity and pass@k performance. Choosing and comparing samplers also requires care, since their relative performance depends on the evaluation protocol: compute matching, temperature, and stochasticity can reverse rankings of re-masking strategies (Shah et al., 2026), while prompt templates and other seemingly minor evaluation choices can affect rankings of parallel decoding methods (Zhang et al., 2026). These findings complicate the attribution of differences in generated-text scores to training recipes alone. To compare recipes as pretraining progresses, our primary downstream measurements therefore avoid multi-step generation. For generated-text metrics, we use the reference sampler associated with each recipe, document its configuration, and calibrate temperature through our entropy-matching protocol (Appendix C.2). These measurements characterize each recipe under its reference sampler. Systematically comparing or optimizing sampling procedures is outside the scope of our training benchmark.

Tracking training progress. Most work on DLM evaluation asks how to assess a trained model, while relatively less attention has been paid to what makes a useful *progress measure*: a signal that can be evaluated repeatedly across checkpoints to determine whether one training procedure is advancing faster than another. This distinction is particularly important for a speedrun. A suitable metric must not only be informative about model quality, but also continue to reflect progress during the relevant stage of training and yield meaningful time-to-target comparisons. Existing scaling analyses suggest why the native training objective is insufficient for this purpose: the relationship between model size, data, compute, and loss depends substantially on the underlying diffusion formulation (von Rütte et al., 2026). Thus, "training speed" across heterogeneous DLM recipes cannot generally be defined simply as the time required to reach a common value of a recipe-specific loss.

Training benchmarks provide a useful methodological precedent. MLPerf and work on benchmarking neural-network training algorithms compare methods under controlled training setups and common performance targets (Mattson et al., 2020; Dahl et al., 2023); the nanoGPT speedrun applies a similar idea to language-model pretraining (Jordan et al., 2024). For autoregressive language models, validation likelihood provides a natural common finish line. For DLMs, however, the lack of an equally accessible and directly comparable sequence-level likelihood makes the choice of finish line itself part of the evaluation problem.

Our work focuses on this preceding question: *which metric should define progress in a DLM pretraining speedrun?* Rather than proposing another general-purpose measure of DLM quality, we audit candidate metrics for the properties required of a common progress target across heterogeneous training recipes. Specifically, we test whether a candidate metric (i) preserves known model orderings

where such a reference is available, (ii) continues to improve throughout early pretraining, and (iii) induces time-to-target recipe rankings that are robust to small perturbations of the target threshold. The resulting HellaSwag speedrun is one concrete instantiation of this procedure at the scale studied here, rather than a claim that HellaSwag is a universal measure of DLM quality.

B Reference DLM training recipes

We train seven recipes: SUBS (Sahoo et al., 2024); SEDD (Lou et al., 2024) and D3PM (Austin et al., 2021) each with masking or uniform noise; and DUO with its original curriculum (Sahoo et al., 2025) or its top-k approximation (Deschenaux et al., 2026). Our implementation ports each objective, noise schedule and sampler from the official code. The anonymous repository given with the submission references the pinned reference commits we used from each official repository, namely the MDLM repository for SUBS, the SEDD repository, the D3PM text code of Austin et al. (2021), and the DUO repository for both DUO variants. Appendix B.2 lists where we depart from them. While the anonymous repository lists all the configurations we used, this section reports the main setup.

B.1 Training setup and hyperparameters

Data. Every run reads the same 3,276,800,000-token prefix of the training shards of `kjj0/fineweb10B-gpt2`, and we use its associated validation set in Appendix D.3. Samples are GPT-2 tokenized and separated by the end-of-text token. Batches are contiguous 1,024-token windows, which may span document boundaries, read in the same fixed order for every recipe and seed. The seed therefore changes the initialization and the training-time noise, not the data.

Optimization. All recipes share the backbone of Section 2, with a 128-dimensional noise-level embedding, and the same optimizer: AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$ and no weight decay, a learning rate of 3×10^{-4} held constant after a linear warmup, gradient clipping at 1.0, no dropout, and antithetic sampling of noise times. Training uses bfloat16 mixed precision and data parallelism over eight GH200 GPUs. Table 1 lists what differs between recipes.

Sampling. Generated-text metrics use each recipe's sampler (Table 1) with 1,000 denoising steps, at the temperature tuned for every checkpoint in Appendix C.2.

Table 1: **Recipe-specific settings.** Tokens per step count all eight GPUs. Warmup is in optimizer steps for the corresponding batch size. Time is the mean elapsed training time over the five seeds to process the fixed stream of approximately 3.3B tokens. Everything else is shared (Appendix B.1). Noise schedules and sampler names are explained below.

	SUBS	SEDD mask	SEDD unif.	D3PM mask	D3PM unif.	DUO original	DUO top-k
Noise	mask	mask	uniform	mask	uniform	uniform	uniform
Tokens per step	524k	164k	205k	262k	205k	205k	205k
Optimizer steps	6,250	20,000	16,000	12,500	16,000	16,000	16,000
Warmup steps	384	2,500	2,500	384	384	384	384
Noise schedule	log-linear	log-linear	geometric	log-linear	linear	linear	linear
Noise-level input	no	yes	yes	yes	yes	yes	yes
Sampler	ancestral	analytic	analytic	ancestral	ancestral	ancestral	ancestral
Time (min)	28.9	39.1	35.7	35.8	69.7	63.4	51.2

Noise schedules and samplers. For the discrete corruption process, let $\alpha(t)$ be the probability of keeping a token rather than replacing it with noise. The schedule names mean

$$\alpha(t) = \begin{cases} \max(1-t, \varepsilon), & \text{linear}, \\ \max(1-(1-\varepsilon)t, \varepsilon), & \text{log-linear}, \\ \exp[-\sigma_{\min}^{1-t}\sigma_{\max}^{t}], & \text{geometric}, \end{cases}$$

where ε is a small numerical floor and the geometric schedule uses $\sigma_{\min} = 10^{-4}$ and $\sigma_{\max} = 20$. An ancestral sampler uses predicted clean-token probabilities in the backward diffusion transition. SEDD's analytic sampler instead constructs that transition from predicted density ratios (Lou et al., 2024). Both progressively remove noise over the chosen sampling steps. DUO's continuous relaxation and curriculum are described next.

B.2 Departures from the official implementations

As explained in Section 2, we start from the nanoGPT speedrun setup and adapt its architecture for DLM. The backbone follows the 12-layer "small" Diffusion Transformer used by MDLM, SEDD, and DUO. We adopt their OpenWebText settings for the optimizer, learning rate, gradient clipping and precision, then tune nearby values for our shorter FineWeb runs. We use no dropout or exponential moving average of the weights, and recipe-specific batch sizes and warmup lengths (Table 1).

All recipes generate samples in 1,000 steps at a tuned temperature. SEDD predicts density ratios rather than token probabilities, so its temperature is applied to relative transition weights. Temperature one recovers its reference analytic sampler.

SUBS and SEDD otherwise follow the official settings, including SEDD's geometric noise from 10^{-4} to 20 with uniform noise. D3PM keeps the original 1,000 discrete steps but not its loss weighting: the original text model minimizes $L_{\mathrm{vb}} + 0.01\, L_{\mathrm{ce}}$, whereas we use $10^{-3} L_{\mathrm{vb}} + L_{\mathrm{ce}}$ with masking and L_{vb} alone with uniform noise. Both DUO variants use the OpenWebText script's noise range $\gamma \in [-3.55, -1.85]$ and softmax temperature 10^{-3}. The temperature stays constant throughout our runs: the script sets both temperature endpoints to 10^{-3}, and its relaxed phase ends at 500k steps, beyond our training budget. Our top-k variant uses the authors' sparse approximation with $k = 32$ and their cached conversion from Gaussian noise to discrete corruption probabilities (`efficient_cached`). This differs from the polynomial approximation (`poly9`) used for the speed experiments in DUO II (Deschenaux et al., 2026). We therefore call this variant DUO top-k, rather than a reproduction of the full DUO II configuration.

B.3 Saved checkpoints

Every recipe is trained five times and saved at the same ten positions, approximately 17M, 101M, 268M, 671M, 1.07B, 1.61B, 2.15B, 2.68B, 3.09B and 3.28B training tokens. The exact count is the first optimizer step at each target and therefore differs slightly with batch size. The first four checkpoints help to track the rapid changes after initialization and place observations around every recipe's learning-rate warmup endpoint, which lies between 79M and 512M tokens. The middle positions are separated linearly by about 0.4–0.54B tokens, and the final gap contracts to 0.19B tokens. This gives more resolution where curves change fastest and near the end of the measured training budget, while keeping the same token budgets for every recipe.

Table 2: **Measured times underlying the stability sweep.** Each entry is the average elapsed training time in minutes over five runs. Columns follow the ten token budgets in Appendix B.3. We linearly interpolate scores between these times to evaluate the 1–27-minute sweep. Displayed times are rounded; calculations use the recorded values.

Recipe	1	2	3	4	5	6	7	8	9	10
SUBS	0.21	0.94	2.41	5.94	9.47	14.18	18.92	23.64	27.17	28.86
SEDD mask	0.25	1.25	3.25	8.05	12.85	19.25	25.68	32.07	36.86	39.15
D3PM mask	0.25	1.16	2.98	7.36	11.74	17.58	23.45	29.29	33.68	35.77
DUO original	0.36	1.98	5.23	13.01	20.79	31.17	41.58	51.95	59.73	63.43
DUO top-k	0.33	1.64	4.25	10.52	16.80	25.17	33.56	41.93	48.20	51.18
SEDD uniform	0.21	1.12	2.94	7.33	11.71	17.56	23.43	29.26	33.64	35.73
D3PM uniform	0.39	2.17	5.73	14.28	22.84	34.24	45.67	57.07	65.63	69.69

B.4 Training and validation losses

The resulting training curves (and initial and final validation losses) are reported in Figure 4.

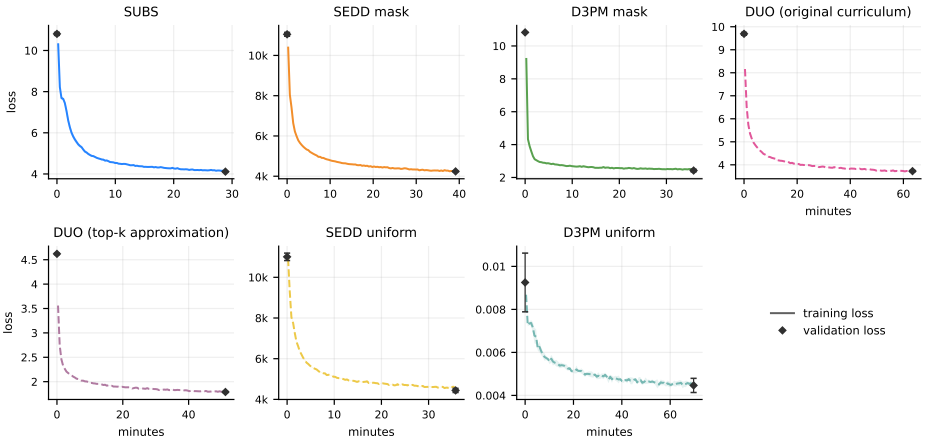


Figure 4: **Training and validation losses of each recipe** over elapsed training time. Curves show each recipe's own training loss (on different vertical scales), averaged over intervals of 1% of the training tokens, then over five runs. Diamonds show validation loss at the start and end of training. Shading and error bars (often narrow) are one standard deviation across runs.

C METRIC FAMILIES

This appendix gives the definitions of the metrics introduced in Section 3. Their audit results are in Appendix E. Table 3 lists the metrics, the direction counted as improvement, and what each metric needs beyond the trained model.

Table 3: **Metrics audited in this paper.** The arrow gives the direction counted as improvement. "Samples" marks metrics computed on text generated by the model associated with the native sampler for the associated recipe. "Finetuning" marks metrics evaluated after adaptation of the model. "External model" names any additional model needed to compute the metric.

Metric	What it measures	Dir.	Samples	Finetuning	External model	Defined in
Downstream tasks						
HellaSwag, ARC-Easy, PIQA (`norm`, `no-norm`)	multiple-choice accuracy from answer reconstruction	↑	–	–	–	Appendix C.1.2
BLiMP	preference for the grammatical sentence	↑	–	–	–	Appendix C.1.2
MNLI, direct	entailment-label accuracy	↑	–	–	–	Appendix C.1.2
LAMBADA	exact recovery of a final word	↑	–	–	–	Appendix C.1.2
MNLI, adapted	entailment-label accuracy	↑	–	✓	–	Appendix C.1.3
Seven synthetic tasks, adapted	exact-match answers	↑	✓	✓	–	Appendix C.1.3
Reference scores						
Gen. PPL, BPB	likelihood of samples under a scorer	↓	✓	–	AR scorer	Equation (2)
Within one sample						
Clean decoded text	samples without malformed characters	↑	✓	–	–	Appendix C.3.1
Compression ratio	compressed / original bytes	↑	✓	–	–	Appendix C.3.1
Unique-word ratio	distinct / total words	↑	✓	–	–	Appendix C.3.1
Trigram, 4-gram repetition	repeated word n-grams	↓	✓	–	–	Appendix C.3.1
Across samples						
Exact duplicate rate	copies within the bank	↓	✓	–	–	Appendix C.3.2
Semantic Vendi	diversity of sample embeddings	↑	✓	–	MPNet	Appendix C.3.2
Self-BLEU-3	n-gram overlap between samples	↓	✓	–	–	Appendix C.3.2
Set compression	compressibility of the joined bank	↑	✓	–	–	Appendix C.3.2
Distinct-3	distinct / total trigrams in the bank	↑	✓	–	–	Appendix C.3.2
Similarity to held-out text						
MAUVE	overlap of embedding distributions	↑	✓	–	GPT-2 Large	Appendix C.4
Unigram, bigram JS	divergence of n-gram frequencies	↓	✓	–	–	Appendix C.4

C.1 Downstream tasks

We describe the datasets and prompts, how tasks are scored without finetuning and with finetuning, and the evaluation cost.

C.1.1 Datasets and prompts

Natural-language tasks. Table 4 lists the natural-language datasets, each pinned to a fixed version (clickable link), with the split we evaluate on. We use every example of that split. Adapted MNLI is trained on the MNLI training split, from which a fixed $1/40$ (about 9.8k examples) is held out for validation.

Table 4: **Downstream datasets.** Token columns report median/maximum GPT-2 token counts; "Total" includes the begin/end-of-text token that starts every sequence, the prompt and one answer. Every sequence fits within the model's 1,024-token context.

Task	What is tested	Split	Examples	Choices	Prompt	Answer	Total
HellaSwag	commonsense (Zellers et al., 2019)	validation	10,042	4	52/105	31/87	88/153
ARC-Easy	grade-school science (Clark et al., 2018)	test	2,376	4–5	23/156	3/28	29/160
ARC-Challenge	grade-school science, challenge set (Clark et al., 2018)	test	1,172	4–5	27/169	6/45	34/172
PIQA	physical plausibility (Bisk et al., 2020)	validation	1,838	2	13/32	15/246	31/256
BLiMP	English grammar (Warstadt et al., 2020)	train (only)	67,000	2	0/0	9/30	10/31
MNLI	textual inference (Williams et al., 2018; Wang et al., 2019)	validation (same genres as training)	9,815	3	45/236	1/1	47/238
LAMBADA	final-word completion (Paperno et al., 2016)	test	5,153	1 word	80/222	1/10	83/225

Prompts and formats. Prompts and answer choices follow the task definitions of the LM Eval Harness at commit `4e7e0d4`. We prepend the beginning-of-text token used by each tokenizer. For our DLMs, GPT-2 and Pythia, this is also the end-of-text token. Llama has a separate beginning-of-text token, which we use instead. Answers following a prompt start with a space, and long prompts are truncated from the left. BLiMP has no prompt: each of the two sentences (e.g. "Susan revealed herself." and "Susan revealed themselves.") is scored as a complete answer, and the model should prefer the grammatical one. Direct and adapted MNLI use the prompt `{premise}\nQuestion: {hypothesis}. True, False or Neither?\nAnswer:`, whose three answers "True", "Neither" and "False" are each one GPT-2 token. For LAMBADA, the context is the passage without its last space-separated word, and the target is that word.

Synthetic tasks. The seven synthetic tasks, in the style of standard copying, recall and reasoning benchmarks (Jelassi et al., 2024; Fu et al., 2023; Arora et al., 2024; Ye et al., 2025a), are: copying (8–64 letters), reversing (8–32 letters), sorting (8–24 integers below 100), addition (two integers below 100,000), associative recall (6–18 key–value pairs), bracket completion (8–32 bracket pairs) and 9×9 Sudoku (22–42 givens). For each task, we generate disjoint training (50k), validation (2k) and test (2k) pools without duplicates, from which the adaptation run draws 4,096 training and 32 validation examples per task. Final answer-generation accuracy uses 32 test examples per task; a separate token-reconstruction check uses 128 (Appendix C.1.3). The associated repository contains the full details for reproduction.

C.1.2 Scoring without finetuning: HellaSwag, ARC, PIQA, BLiMP, MNLI direct, LAMBADA

Each multiple-choice question consists of a context c and possible answers $a_0, \ldots, a_J$, one of which is correct (LAMBADA, which is not multiple choice, is treated at the end). For every candidate, we keep the context clean, corrupt the answer with the recipe's noise, and measure how well the model recovers the clean answer. The candidate with the smallest reconstruction score is selected, and we report the fraction of examples for which it is the correct one. The tasks differ only in how the answer is *corrupted* and *normalized*. HellaSwag, ARC and PIQA corrupt the answer *partially*, at many noise levels ("Partial corruption" below). BLiMP, direct MNLI and LAMBADA corrupt the *whole* answer at once ("Full corruption" below).

Normalization options. We consider the two conventions of the LM Eval Harness: `norm` first divides each score by the answer's character count (`acc_norm`), and `no-norm` does not apply such

a normalization (`acc` in LM Eval Harness conventions). For BLiMP and direct MNLI we report only `no-norm`, as the Harness does: BLiMP compares two almost identical sentences, and dividing by character count would favour the longest MNLI label ("Neither").

Partial corruption: HellaSwag, ARC and PIQA. For candidate answer a_j and context c, the clean sequence is $x^{(j)} = (c, a_j)$. On draw k, the recipe's noise level $t_{j,k}$ and corruption rule determine a subset $M_{j,k}$ of answer positions. We corrupt these positions with the recipe's noise to obtain

$$z^{(j,k)} = (c, \widetilde{a}_j^{(k)}), \qquad \widetilde{a}_{j,i}^{(k)} = \begin{cases} \eta_{j,k,i}, & i \in M_{j,k}, \\ a_{j,i}, & i \notin M_{j,k}, \end{cases}$$

where $\eta_{j,k,i}$ is the recipe's noise at that position (the mask token, or a uniformly drawn token). The model receives $z^{(j,k)}$ and the noise level, and we compute the recipe's own training loss on the answer tokens only. The $K = 32$ random corruptions are uniformly spread across all levels of corruption. Averaging these losses gives a raw score $\widehat{L}_j$ for the clean answer a_j. In the notation of Equation (1), each loss receives the clean target a_j, the input $(c, \omega_k(a_j))$, and its noise level; the decision score is $\widehat{s}(a_j) = \widehat{L}_j / d(a_j)$. This reconstruction score plays the role of the Harness likelihood score for a DLM (Nie et al., 2025; Sahoo et al., 2024). Figure 5 illustrates the procedure for SUBS. For each question, we reuse the same random-number seeds when scoring its possible answers, so they are compared under matched corruption.

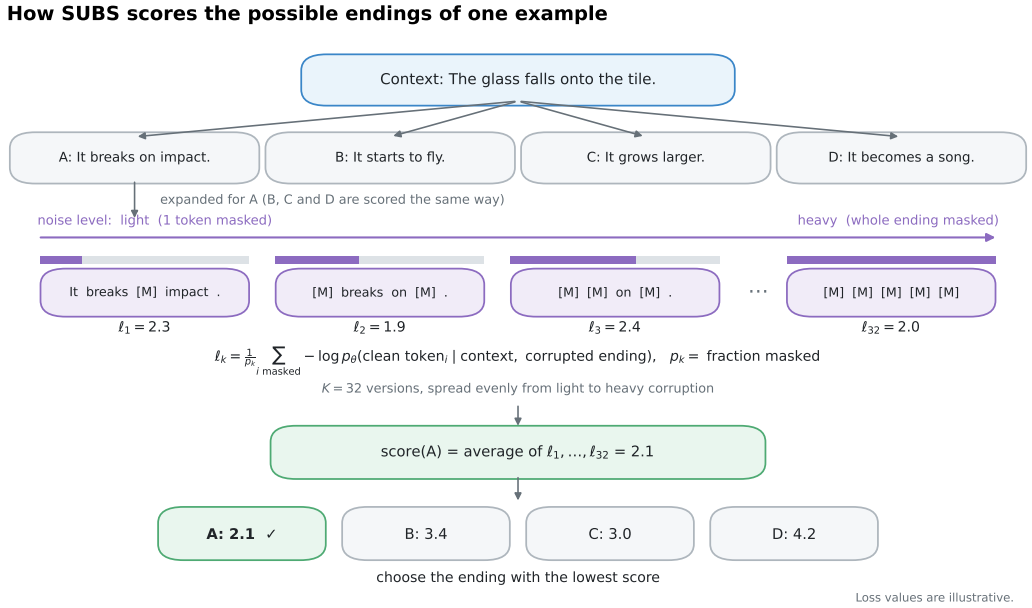


Figure 5: **How SUBS scores the possible endings of one example.** The context stays visible and only the ending is corrupted. Each ending is masked at $K = 32$ noise levels, spread evenly from one masked token to the whole ending (shown for ending A; purple bars give the masked fraction p_k). Each version gives a loss ℓ_k, the recipe's loss on the masked tokens, reweighted by $1/p_k$. The ending's score is the average of the 32 losses, and the ending with the lowest score is selected. Loss values are illustrative.

Worked SUBS example. For SUBS, $\eta_{j,k,i}$ is the mask token m. At noise time t, a clean token is kept with probability α_t and masked otherwise:

$$q_t(z_{t,i} \mid x_i) = \alpha_t \, \mathbf{1}\{z_{t,i} = x_i\} + (1 - \alpha_t) \, \mathbf{1}\{z_{t,i} = m\}, \tag{3}$$

with $\alpha_t = 1 - (1 - \varepsilon)t$ and $\varepsilon = 10^{-3}$. Writing $\ell_\theta(x_i; z_t, \sigma_t) = -\log p_\theta(x_i \mid z_t, \sigma_t)$ with $\sigma_t = -\log \alpha_t$, the continuous-time training objective of SUBS/MDLM (Sahoo et al., 2024) is

$$\mathcal{L}_{\text{SUBS}}(x) = \mathbb{E}_{\substack{t \sim \text{Unif}[\varepsilon, 1] \\ z_t \sim q_t(\cdot \mid x)}} \left[\frac{-\dot{\alpha}_t}{1 - \alpha_t} \frac{1}{n} \sum_{i : z_{t,i} = m} \ell_\theta(x_i; z_t, \sigma_t) \right], \tag{4}$$

where $-\dot{\alpha}_t/(1-\alpha_t) = 1/t$ for this schedule: the loss asks the model to recover the hidden clean tokens, with a weight that compensates for how many tokens are hidden. Let n_j be the number of tokens of a_j. On draw k, we choose a number of masked tokens $r_{j,k} \in \{1, \dots, n_j\}$ and mask a uniformly chosen subset $M_{j,k}$ of that size. The masked fraction and model input are $p_{j,k} = r_{j,k}/n_j$, $\alpha_{j,k} = \max(1 - p_{j,k}, \varepsilon)$ and $\sigma_{j,k} = -\log \alpha_{j,k}$; SUBS ignores $\sigma_{j,k}$, following MDLM. The counts $r_{j,k}$ are spread evenly: we split the 32 draws into two blocks of 16, $k = 16b + k'$ with $b \in \{0, 1\}$ and $k' \in \{0, \dots, 15\}$, choose a random start $s_{j,b} \in \{1, \dots, n_j\}$ for each block, and set

$$r_{j,k} = 1 + \left(\operatorname{round}\left(s_{j,b} + \frac{k' n_j}{16} \right) - 1 \right) \bmod n_j.$$

The SUBS answer score is

$$\widehat{L}_j = \frac{1}{32} \sum_{k=0}^{31} \frac{1}{p_{j,k}} \sum_{i \in M_{j,k}} -\log p_\theta \left(a_{j,i} \mid z^{(j,k)}, \sigma_{j,k} \right),$$

where the factor $1/p_{j,k}$ compensates for observing only the masked subset.

Other recipes. For masked SEDD and D3PM, a draw first selects a noise time from the recipe's schedule and then masks each answer token independently with that time's probability, so a draw may leave the answer unchanged. Uniform-noise SEDD and D3PM instead replace the selected tokens by tokens drawn uniformly from the vocabulary:

$$q_t(z_{t,i} \mid x_i) = \alpha_t \, \mathbf{1}\{z_{t,i} = x_i\} + (1 - \alpha_t) \frac{1}{|V|}. \tag{5}$$

DUO uses its own relaxed uniform forward process.

Full corruption: BLiMP, direct MNLI and LAMBADA. We replace every answer token by a mask or a uniformly drawn token, leaving the prompt unchanged. For models that take the noise level as an input, we give a near-fully-corrupted noise level, using a keep-probability setting of 10^{-3}, or 10^{-12} for D3PM. For BLiMP and direct MNLI, we score each corrupted answer in one forward pass and average the scores:

$$\widehat{s}(a_j) = \frac{1}{D} \sum_{d=1}^{D} \sum_{i=1}^{n_j} -\log p_\theta \left(a_{j,i} \mid c, \widetilde{a}_j^{(d)} \right),$$

where $\widetilde{a}_j^{(d)}$ is the fully corrupted answer and p_θ is the recipe's prediction of the clean token at the noise level just described. For mask recipes the corrupted answer is always the same, so $D = 1$. For uniform recipes it is redrawn and $D = 16$. The candidate with the smallest $\widehat{s}$ is selected.

LAMBADA asks for the last word of a passage, which can only be guessed from the broader context, without prescribed multiple choices. We corrupt the target word as above, then take the most likely token at every target position in a single denoising step (for uniform recipes, after averaging the predicted probabilities over 16 corruptions), and count the example as correct only if all of them match, i.e. the complete word is recovered.

C.1.3 Scoring with finetuning: MNLI adapted, synthetic tasks

MNLI, adapted. We finetune each checkpoint with the multiple-choice recipe of Radford et al. (2018): each candidate (the direct-MNLI prompt and one label, Appendix C.1.1, followed by an end-of-text token) is read uncorrupted, and a linear head on that token's final hidden state gives its score, trained with cross-entropy over the three candidates. We train on the MNLI training split and validate on a held-out part of it (Appendix C.1.1; AdamW, learning rates 10^{-3} for the head and 10^{-5} for the backbone, weight decay 0.01, batch size 8, at most 2,048 updates), evaluate the validation cross-entropy every 128 updates, and stop after four evaluations without improvement. The best checkpoint is evaluated once on the MNLI validation split with the same genres as training (Table 4).

Synthetic tasks, adapted: Copying, reversing, associative recall, bracket completion, sorting, addition and Sudoku. The seven synthetic tasks (Appendix C.1.1) have near-zero direct accuracy, so we evaluate them after adaptation. For each checkpoint, one run finetunes all parameters on 4,096

pooled examples per task, with the recipe's diffusion loss on the answer tokens and the prompt visible (AdamW, learning rate 10^{-4}, batch size 16). Every 64 updates, we hide the answers of 32 validation examples per task and measure how well the model predicts their tokens. For uniform noise, we average probabilities over 16 random replacements before computing the loss. We use this validation score to select the checkpoint. Final accuracy is measured separately by generating answers with 64 denoising steps.

C.1.4 Evaluation cost

Table 5 reports the cost of evaluating one checkpoint on each task, in GH200 GPU-hours.

Table 5: **Evaluation cost per checkpoint** on each complete task split, in GH200 GPU-hours, averaged over five trainings and ten checkpoints per recipe. Costs include model loading, so they scale only approximately with the number of corruptions.

Task	Corruptions mask / uniform	GPU-h per corruption mask / uniform	Total GPU-h
HellaSwag	32 / 32	0.00956 / 0.02842	0.6508
BLiMP	1 / 16	0.02278 / 0.02096	0.2014
ARC-Easy	32 / 32	0.00213 / 0.00673	0.1523
PIQA	32 / 32	0.00115 / 0.00260	0.0633
MNLI, direct	1 / 16	0.00726 / 0.00547	0.0532
LAMBADA	1 / 16	–	0.0185
Seven synthetic tasks, jointly adapted	–	–	0.0772
MNLI, adapted	–	–	0.0736

C.2 Reference scores: Gen. PPL and BPB

Scorers. The six scorers are GPT-2 Large and GPT-2 XL, GPT-Neo 2.7B (Black et al., 2021), Pythia 1B, Qwen2.5 7B (Qwen Team, 2024), and Qwen3 1.7B Base (Qwen Team, 2025). They cover four model families, sizes from 0.8B to 7B parameters, and several tokenizers and training corpora. Every scorer grades the same text, cut to at most 1,023 GPT-2 tokens, which fits within all their context windows.

Entropy matching. We match the entropy target of Section 3.2 by bisection over temperatures in $[0.003, 1.3]$, on banks of 64 samples for DLMs and 128 for AR models. Each sample is cut to the longest string that fits 1,023 GPT-2 tokens, so all six scorers grade the same text. The search succeeded for all 350 audited checkpoints. For illustration, we include Figures 6 and 7, which use a coarser sweep (eleven temperatures from 0.003 to 1.3, eight samples each, at all saved checkpoints per run).

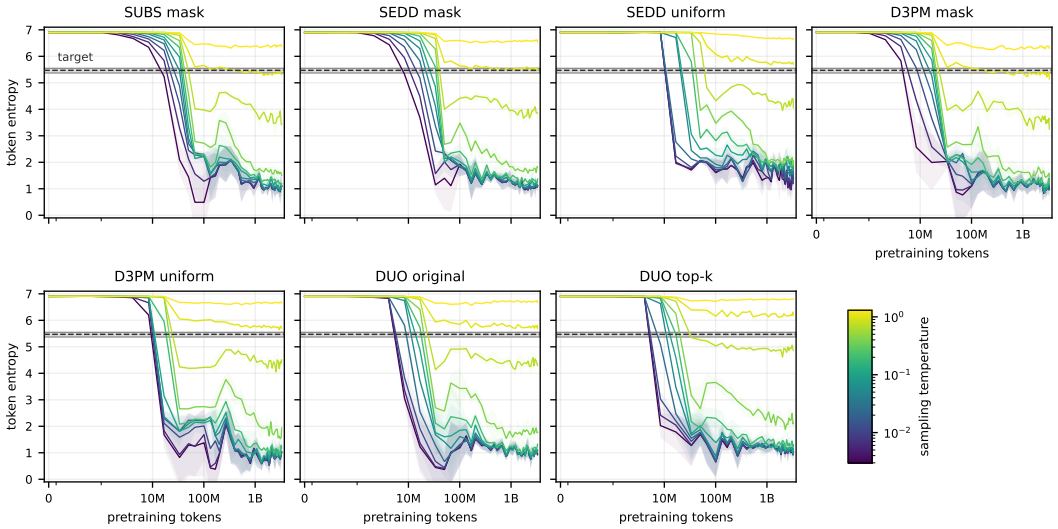


Figure 6: **The full temperature sweep along each recipe.** Each color fixes one temperature; lines are means over five training runs and shading is one standard deviation. The dashed line marks the 5.471 target, and the gray band the interquartile range of held-out OpenWebText (5.37–5.55).

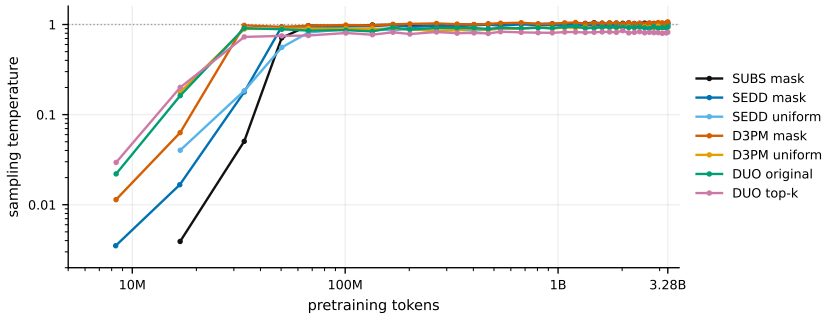


Figure 7: **The inverse view of the same sweep:** the sampling temperature needed to reach token entropy 5.471, interpolated from the sweep of Figure 6. A curve starts only once the sweep brackets the target (at the earliest checkpoints even the lowest temperature gives higher entropy). Curves average the five training runs (dots mark saved checkpoints). The standard deviation across runs is at most 8% of the mean (typically 2%), too small to show cleanly.

C.3 Diversity within and across samples

Metrics in this section are computed on the entropy-matched bank S of each checkpoint, the same samples the reference scorers grade (Appendix C.2): 64 samples, cut to at most 1,023 GPT-2 tokens.

C.3.1 Within one sample

These metrics flag two ways a sampler can fail: malformed output, and repetition within a sample, measured through repeated word n-grams and compressibility. We compute each metric for one sample, then average over the samples of the bank.

Clean decoded text. When generated token bytes cannot form valid UTF-8, decoding inserts the replacement character U+FFFD. We count this character and unwanted ASCII controls (U+0000–U+0008, U+000B, U+000C, U+000E–U+001F and U+007F). Let $b(x)$ be their total count in sample x. The fraction of clean samples is

$$V_{\text{clean}} = \frac{1}{|S|} \sum_{x \in S} \mathbf{1}\{b(x) = 0\}. \tag{6}$$

Compression ratio. Repeated text is easier to compress. We compress the decoded sample with `zlib` at level 9 and divide its size by the original UTF-8 byte count: $C(x) =$ compressed bytes/original bytes.

Word n-gram repetition. For this metric, we split decoded text at spaces, tabs and line breaks, ignore capitalization, and keep punctuation attached. We call these pieces *words*: they are not the model's tokens. For example, `Cat` and `cat` count as the same word, but `cat` and `cat,` differ. Semantic Vendi and MAUVE instead use the tokenizers of their respective representation models (Appendix C.3.2 and Appendix C.4).

An n-gram is a group of n consecutive words, with overlapping groups counted separately. We measure the fraction of occurrences that repeat an earlier group in the same sample:

$$R_n(x) = 1 - \frac{\#\{\text{distinct contiguous } n\text{-grams}\}}{\#\{\text{contiguous } n\text{-gram occurrences}\}}. \tag{7}$$

In particular, the case $n = 1$ gives the unique-word ratio used in the table: $U(x) = 1 - R_1(x)$. For example, `The cat sat the cat sat` has six words and three different ones after ignoring capitalization, so $U(x) = 3/6 = 0.5$. There are also four trigrams: `the cat sat`, `cat sat the`, `sat the cat`, and `the cat sat`. Three are different, so $R_3(x) = 1 - 3/4 = 0.25$.

C.3.2 Across samples

These metrics measure how much the samples of the bank S overlap, without a held-out reference bank: exactly (duplicates), in their words (Self-BLEU, Distinct-3 and set compression) or in meaning (semantic Vendi).

Exact duplicate rate. We count how many samples are copies of a sample already in the bank: $D_{\text{exact}} = 1 - |\operatorname{unique}(S)|/|S|$. Here we compare the decoded strings exactly, keeping capitalization and whitespace.

Semantic Vendi. We obtain one vector per sample with the `all-mpnet-base-v2` model. It reads up to 512 of its own tokens. We average their final representations, excluding padding, and normalize the vector to length one. This metric asks how varied the samples are (it does not involve a comparison with a held-out distribution). Let K_{ij} be the cosine similarity of the `all-mpnet-base-v2` embeddings of samples i and j. If λ_j are the eigenvalues of $K/|S|$, the score is

$$\operatorname{Vendi}(K) = \exp\left(- \sum_{j:\lambda_j > 0} \lambda_j \log \lambda_j \right). \tag{8}$$

A larger value means that the sample vectors are more varied (Friedman & Dieng, 2023).

Self-BLEU-3. We compare each sample with all others using BLEU-3 (Papineni et al., 2002) on the words defined above, then average: $\operatorname{SelfBLEU}_3(S) = |S|^{-1} \sum_{x \in S} \operatorname{BLEU}_3(x, S \setminus \{x\})$. A

larger value means more overlap between samples. We use equal weights for orders 1–3 and NLTK smoothing method 1, matching the Texygen convention (Zhu et al., 2018).

Set compression. We join the 64 samples with a newline, the literal separator `<|sample_boundary|>`, and another newline. We compress this combined text with `zlib` at level 9 and divide by its original UTF-8 byte count. We average this ratio over ten sample orders drawn with seed 1729. Compression can exploit patterns shared by different samples, not just repetition within one.

Distinct-3. We pool all within-sample word trigrams and count the fraction that are distinct:

$$\mathrm{Distinct}_3(S) = \frac{\#\{\text{distinct trigrams across } S\}}{\#\{\text{trigram occurrences across } S\}}. \tag{9}$$

Unlike trigram repetition in Appendix C.3.1, this also detects a trigram repeated in different samples.

C.4 Similarity to held-out text

These metrics compare the generated bank with held-out FineWeb text from a validation shard disjoint from training: MAUVE in the space of an embedding model, and unigram and bigram Jensen–Shannon divergence through word and word-pair frequencies.

MAUVE. MAUVE groups generated and held-out samples using their text representations, then compares how often the two sets occupy each group (Pillutla et al., 2021). Larger values mean closer agreement. At every DLM checkpoint and for every AR model, we compare 1,000 generated samples with the same 1,000 held-out FineWeb samples. We use the reference implementation's GPT-2 Large final-token representations, automatic bucket count (100 here), and average clustering seeds 25, 26 and 27. We also retain its smoothed variant, MAUVE-star (Liu et al., 2021; Pillutla et al., 2023), in the released measurements and AR table. The three clustering runs reuse the same samples so their variations do not measure the uncertainty from generating another bank.

Sampling and entropy. Each DLM uses its recipe's sampler with 1,000 denoising steps, without top-k truncation. We tune temperature toward entropy 5.471, then generate a separate evaluation bank, retaining its first 1,000 samples without filtering on entropy. Samples are limited to 1,023 GPT-2 text tokens after removing the beginning-of-text token. The tolerance ± 0.02 is met by 289 of 350 DLM banks: 36/50 for SUBS, 41/50 for SEDD mask, 43/50 for SEDD uniform, 50/50 for each D3PM variant, 33/50 for DUO original and 36/50 for DUO top-k. The other 61 banks have a median absolute gap which remains moderate: 0.030, with a maximum of 0.082. We do not tune the temperature at a finer resolution for computational reasons. The AR bank entropies are reported in Table 11.

Unigram and bigram Jensen–Shannon divergence. We count individual words or consecutive word pairs across each bank. As in Appendix C.3.1, we split at spaces, tabs and line breaks, ignore capitalization, and keep punctuation attached. If P and Q are the resulting frequency distributions in generated and held-out text, respectively, then

$$\mathrm{JS}(P,Q) = \tfrac{1}{2} D_{\mathrm{KL}}(P\|M) + \tfrac{1}{2} D_{\mathrm{KL}}(Q\|M), \qquad M = \tfrac{1}{2}(P+Q). \tag{10}$$

We use base-2 logarithms and no smoothing. Zero means that the frequencies agree. Bigrams never cross sample boundaries. MAUVE and both JS metrics use the same sample banks.

D Metric screening protocol

We describe the metric audit through three checks, in the order used in the main text: stability of elapsed-time recipe rankings when the target changes (Appendix D.1); improvement between adjacent DLM checkpoints (Appendix D.2); and recovery of autoregressive rankings measured from likelihood (Appendix D.3).

D.1 Stability: robustness to small changes in the target

We ask whether moving the finish line a little changes which recipe reaches it first (Section 4.1). At each training budget, we set the target to the best recipe score and try nearby targets. We report how often recipe pairs change order, together with how many pairs the saved runs let us compare.

Target at a training budget. For each recipe r, we average the scores and elapsed times of its five training runs at each of the ten saved checkpoints, and interpolate successive averages linearly into a curve $f_r(t)$. For a metric where smaller is better, we flip its sign so that larger is better. The best score at time t is $g(t) = \max_r f_r(t)$, and the target at budget b is $q_b = g(b)$. We use whole-minute budgets at which all seven recipes have observations covering $\pm 5\%$ around b, which gives $b \in \{1, \ldots, 27\}$ for every metric. The interpolation knots are the mean elapsed times in Table 2.

Nearby targets. We move the target within $J_b = [q_b - A_b - B_b,\ q_b + A_b + B_b]$, where

$$A_b = \sqrt{\mathbb{E}_t \sum_{r=1}^{7} \frac{v_r(t)}{5}}, \qquad B_b = \sqrt{\mathbb{E}_t \left[g(t) - q_b\right]^2}, \tag{11}$$

t is uniform over $[0.95b, 1.05b]$, and $v_r(t)$ is the sample variance of the five runs of recipe r at time t. A_b is the seed-to-seed variation of a recipe average (hence the division by five), pooled over the seven recipes; B_b is how much the best score changes when the budget moves by $\pm 5\%$. We clip J_b only at the metric's bounds, such as 0 and 1 for accuracy. Table 6 gives their size for four example metrics.

Table 6: **How the local target ranges are set for four examples, at five training budgets.** Task accuracies and their perturbations are in percentage points. BPB is in bits per byte. A summarizes variation across trainings and B a $\pm 5\%$ time change (Equation (11)); the target range is the target $\pm(A + B)$. These are perturbation ranges, not confidence intervals. The reproduction files give these values for every metric.

		Budget (minutes)				
Metric		1	5	10	20	27
HellaSwag norm	Target	25.828	25.863	26.931	28.045	28.745
	A	0.167	0.214	0.256	0.225	0.238
	B	0.003	0.028	0.043	0.053	0.062
LAMBADA	Target	0.000	0.393	2.927	5.505	7.092
	A	0.000	0.166	0.750	1.160	1.498
	B	0.000	0.022	0.194	0.084	0.225
ARC-Easy no-norm	Target	26.631	30.110	31.536	32.872	33.734
	A	0.349	0.672	0.806	0.863	1.126
	B	0.022	0.053	0.048	0.073	0.144
Qwen3 1.7B Base BPB	Target	2.231	1.694	1.604	1.568	1.506
	A	0.067	0.083	0.069	0.050	0.043
	B	0.004	0.004	0.000	0.005	0.000

Comparing recipe pairs. We compare two recipes only when the saved runs tell us which reaches the target first, both at the original target and at the shifted target.

At each budget, we test the 21 recipe pairs at 2,001 equally spaced targets in J_b. We count a pair separately at each target and compute

$$U_b = \frac{\text{changed comparisons}}{\text{comparable comparisons}}, \qquad C_b = \frac{\text{comparable comparisons}}{21 \times 2{,}001}. \tag{12}$$

We report $100U_b$ as *order changes* and $100C_b$ as *coverage*. The averages over time in Equation (11) also use 2,001 equally spaced points.[1] We give no change rate when no comparison is possible, and no stability score for constant metrics. An asterisk in the tables marks targets already reached at the first saved checkpoint, whose earlier crossing we cannot locate.

D.2 Monotonicity: improvement across shared checkpoints

We evaluate every metric at the ten shared checkpoints of Appendix B.3. At each position, we first average the five trainings of a recipe and then compare its nine adjacent values.

[1] Recipes reaching a target at exactly the same time count as tied; a change to or from such a tie counts as an order change. Every counted change in our results is a strict reversal.

For each metric, let $s = 1$ when a larger value is better and $s = -1$ when a smaller value is better. For recipe r, the improvement fraction for average measurements $\bar{y}_{r,j}$ at the ten checkpoints $j = 1, \ldots, 10$ is

$$I_r = \frac{1}{9} \sum_{j=1}^{9} \mathbf{1}\{s(\bar{y}_{r,j+1} - \bar{y}_{r,j}) > 0\}, \qquad I = \frac{1}{7} \sum_{r=1}^{7} I_r. \tag{13}$$

The macro-average I gives every recipe equal weight. Thus, $I = 1$ means that all seven mean trajectories improve at every adjacent pair. There are 63 comparisons in total, so one additional improving pair changes $100I$ by about 1.6 percentage points.

D.3 Recovery: autoregressive calibration

Reference text. The reference text used to rank the autoregressive models by likelihood is the validation shard of the public `kjj0/fineweb10B-gpt2` dataset (Penedo et al., 2024). This shard is disjoint from the shards used to train our DLMs. We take the first 128 distinct samples containing at least 384 GPT-2 tokens and retain at most 1,023 GPT-2 tokens per sample. Each model scores the same retained text with its own tokenizer, so its token count can differ. We prepend the model's beginning-of-text token (BOS), which is also its end-of-text token for GPT-2 and Pythia, but not for Llama. This prefix lets a causal model score the first text token and is itself excluded from the loss. The downstream evaluations use the same prefix convention.

Reference order. If $L_m(x)$ is model m's negative log-likelihood for sample x and $N_m(x)$ its number of scored tokens, we order the models of each family by $\sum_x L_m(x) / \sum_x N_m(x)$ (pooled over tokens, not averaged per sample). The measured orders are in Table 7. We omit Llama 3.1 405B for computational reasons. The orders are not equally separated. For two successive models with average negative log-likelihood $\ell_1 > \ell_2$, we report the relative percentage decrease $100G$ in negative log-likelihood where

$$G = \frac{\ell_1 - \ell_2}{\ell_1}. \tag{14}$$

To show uncertainty in these gaps, we draw 128 samples with replacement 20,000 times, using the same drawn samples for both models, and report the 2.5–97.5 percentile interval.

Table 7: **FineWeb loss gaps for the AR orders to recover.** Lower loss is better. The relative decrease is defined in Equation (14). Intervals come from 20,000 joint resamplings of the same 128 samples. Absolute losses are comparable within a family, whose models share a tokenizer. Each sample starts with the model's beginning-of-text token, which is not included in the loss.

Family	Pair, lower loss on the right	NLL/token left → right	Relative decrease	95% interval for decrease
GPT-2	Small → Medium	3.347 → 3.069	8.30%	[8.01, 8.60]%
	Medium → Large	3.069 → 2.919	4.88%	[4.65, 5.11]%
	Large → XL	2.919 → 2.830	3.06%	[2.84, 3.28]%
Pythia	70M → 160M	5.536 → 3.883	29.86%	[29.38, 30.33]%
	160M → 410M	3.883 → 2.993	22.92%	[22.49, 23.36]%
	410M → 1B	2.993 → 2.811	6.07%	[5.80, 6.35]%
	1B → 2.8B	2.811 → 2.611	7.10%	[6.80, 7.41]%
Llama 2	7B → 13B	1.966 → 1.897	3.54%	[3.27, 3.81]%
	13B → 70B	1.897 → 1.745	8.01%	[6.92, 9.31]%
Llama 3.1	8B → 70B	2.244 → 1.902	15.24%	[12.48, 18.44]%

Auditing metrics with the AR calibration criterion. A metric passes this check when it never reverses an adjacent pair in the measured FineWeb order. For direct downstream tasks, AR models use their own predictions on the same task examples (Appendix C.1 and Table 9). For Gen. PPL, BPB, MAUVE and JS, they instead generate samples that these metrics evaluate (Appendix E.3). We do not apply this check to finetuned tasks or to diversity alone, which aims at measuring a fundamentally different property from likelihood.

E AUDIT RESULTS BY CRITERION

This appendix gives the evidence behind the audit of Section 4, organized by the three checks defined in Appendix D. All results use the seven recipes, five training runs and ten shared checkpoints of Appendix B.3. Appendix E.1 reports the order changes and coverage of every metric over the 1–27 minute sweep. Appendix E.2 reports how often each metric improves between adjacent checkpoints, together with its values over training. Appendix E.3 reports which autoregressive orders each metric family reverses. Finally, Appendix E.4 compares the recipe rankings induced by different metrics.

E.1 STABILITY: ORDER CHANGES AND COVERAGE OF EVERY METRIC

Figure 8 shows both quantities at every minute. The heatmap shows that HellaSwag `norm` has at most 0.1% changes at 71–83% coverage at every minute from 10 to 20. It also shows the regions of few changes and broad coverage of ARC-Easy and PIQA, for instance ARC-Easy `norm` from 4 to 9 minutes (no changes, 71–81% coverage) and PIQA `norm` from 10 to 16 minutes (at most 0.6% changes, 68–71% coverage).

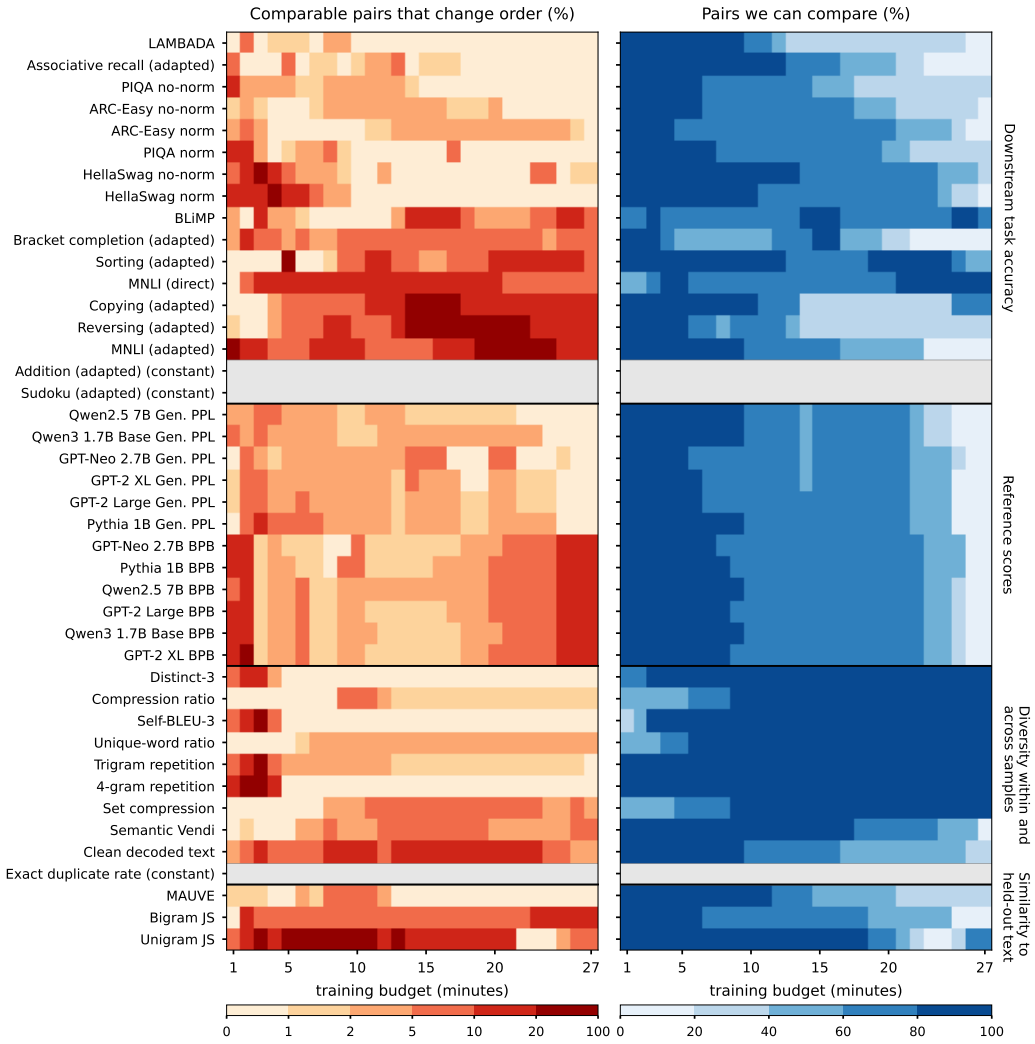


Figure 8: **Local stability of every metric at every minute of the 1–27 minute sweep.** Left: percentage of comparable recipe pairs whose order changes under nearby targets. Right: percentage of the 21 recipe pairs that can be compared (coverage). Rows are grouped by metric family and sorted by average changes within each family; grey rows are constant metrics. Definitions are in Appendix D.1.

E.2 MONOTONICITY: IMPROVEMENT OF EVERY METRIC OVER TRAINING

Table 8 gives, for every metric, the percentage of adjacent checkpoint pairs at which the five-run mean improves (Appendix D.2). These are the percentages quoted in Section 4.2.

We also show the ten checkpoint averages for each metric. In these figures, solid lines denote mask-noise recipes and dashed lines uniform-based recipes. Three metrics are constant and never improve: Addition and Sudoku (adapted) stay at zero accuracy, and the exact duplicate rate stays at zero. They appear at 0% in the table and are therefore omitted from these figures below.

In Figure 9, the tasks with low percentages saturate early (Copying, Reversing), stay near zero (Sorting, Bracket completion) or fluctuate (BLiMP, direct MNLI); LAMBADA stays at zero for SEDD uniform. Figures 10 and 12 show the steady improvement of the reference scores and of similarity to held-out text. In Figure 11, most diversity measures move away from their preferred direction after the first checkpoints, while repetition stays below 1% of trigrams; clean decoded text plateaus early and Semantic Vendi keeps increasing.

Table 8: **Monotonicity of every metric:** percentage of adjacent checkpoint pairs at which the five-run mean improves in the preferred direction, pooled over the seven recipes (63 pairs; Equation (13)). Metrics are grouped by the four metric families and sorted by this percentage within each family.

Downstream task accuracy		*Reference scores*		*Diversity within and across samples*	
ARC-Easy norm	81	GPT-2 Large Gen. PPL	79	Semantic Vendi	81
MNLI (adapted)	81	GPT-2 XL Gen. PPL	79	Clean decoded text	68
ARC-Easy no-norm	79	GPT-Neo 2.7B Gen. PPL	79	Set compression	51
HellaSwag norm	76	Pythia 1B Gen. PPL	79	Self-BLEU-3	46
HellaSwag no-norm	75	Qwen2.5 7B Gen. PPL	76	Distinct-3	41
PIQA norm	70	Qwen3 1.7B Base Gen. PPL	76	Unique-word ratio	41
LAMBADA	67	GPT-2 Large BPB	73	Compression ratio	40
PIQA no-norm	67	GPT-2 XL BPB	73	4-gram repetition	33
Reversing (adapted)	52	GPT-Neo 2.7B BPB	73	Trigram repetition	33
Associative recall (adapted)	51	Pythia 1B BPB	73	Exact duplicate rate (constant)	0
BLiMP	51	Qwen2.5 7B BPB	71		
Copying (adapted)	51	Qwen3 1.7B Base BPB	71	*Similarity to held-out text*	
Bracket completion (adapted)	41			MAUVE	83
MNLI (direct)	33			Bigram JS	73
Sorting (adapted)	24			Unigram JS	70
Addition (adapted) (constant)	0				
Sudoku (adapted) (constant)	0				

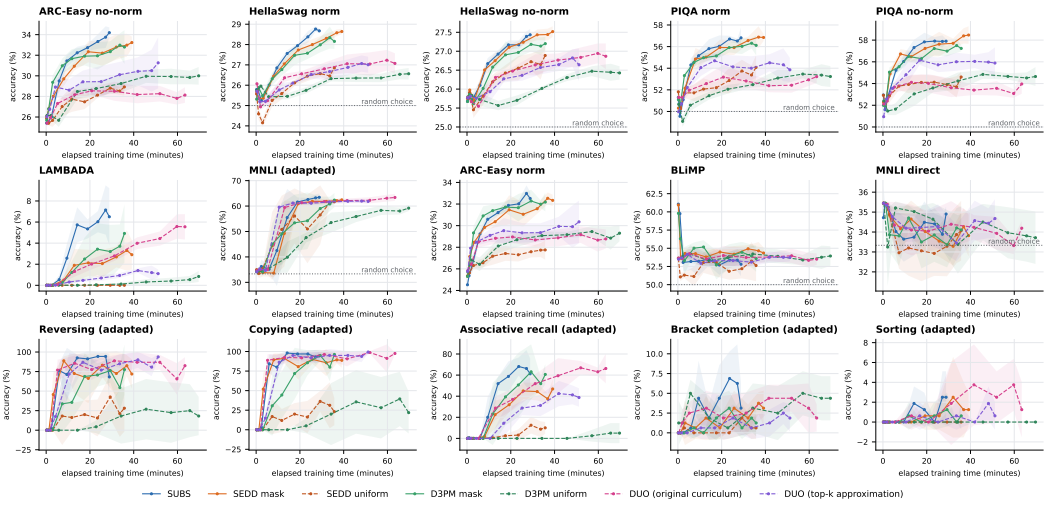


Figure 9: **Downstream accuracy over elapsed training time.** Lines are means over five training runs, dots mark the ten shared checkpoints (the first after 17M tokens), and shading is one standard deviation. Addition and Sudoku (adapted) stay at zero accuracy and are not shown.

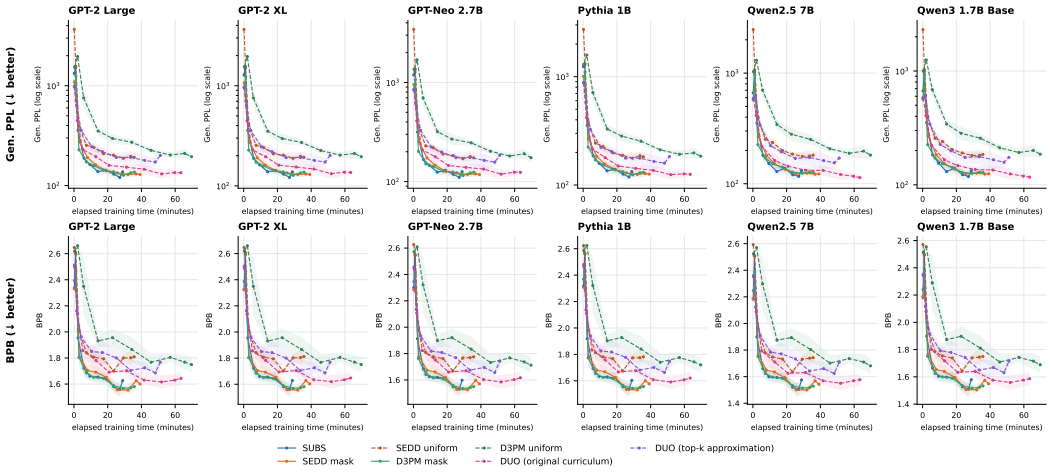


Figure 10: **Reference scores over elapsed training time.** Each column is one of the six scorers, with Gen. PPL (top, log scale) and BPB (bottom) on the entropy-matched sample banks. Lines are means over five training runs, dots mark the ten shared checkpoints, and shading is one standard deviation.

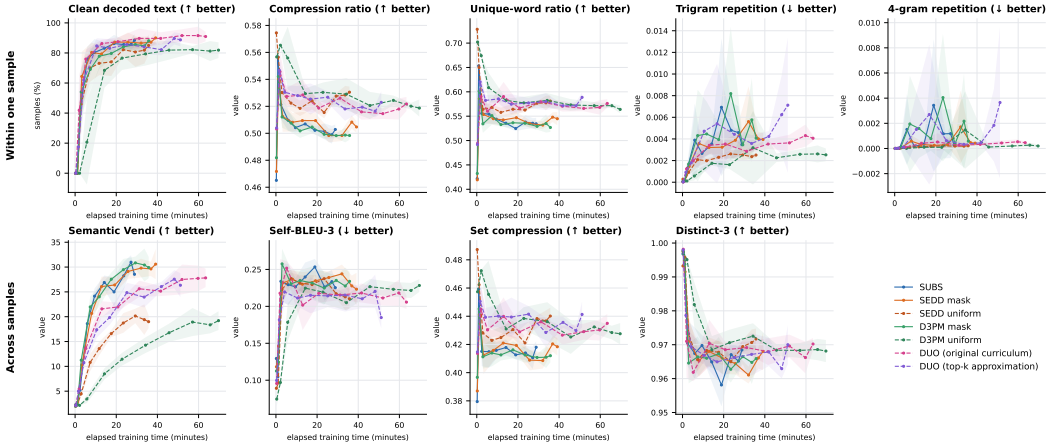


Figure 11: **Diversity within and across samples over elapsed training time.** Lines are means over five training runs, dots mark the ten shared checkpoints, and shading is one standard deviation. Arrows give the direction counted as improvement (Table 3). The exact duplicate rate is zero throughout and is not shown.

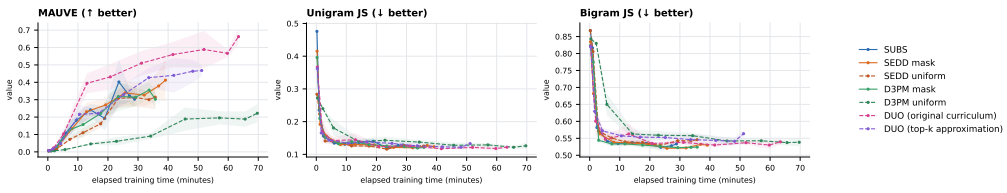


Figure 12: **Similarity to held-out text over elapsed training time.** Lines are means over five training runs, dots mark the ten shared checkpoints, and shading is one standard deviation.

E.3 Recovery: autoregressive orders reversed by each metric family

Downstream task accuracy. Table 9 gives the autoregressive accuracies behind this check. Within each family, rows follow improving FineWeb likelihood and red arrows mark inversions.

We note that the AR pairs have different likelihood gaps, so some orders may be harder to recover than others. The relative decrease in FineWeb negative log-likelihood is smallest for GPT-2 Large/XL (3.1%) and Llama 2 7B/13B (3.5%), compared with 15.2% for Llama 3.1 8B/70B and 22.9% for Pythia 160M/410M (Table 7). BLiMP reverses the two closest pairs, among others, whereas ARC-Easy `norm` and direct MNLI also reverse well-separated models.

Table 9: **Direct downstream accuracy used for autoregressive calibration.** Rows follow improving FineWeb likelihood within each family. Red arrows mark a lower accuracy than the preceding model. The loss gap for each pair is in Table 7. Every model reads the same task examples with its beginning-of-text prefix. BLiMP and MNLI use total scores, and LAMBADA uses exact match.

Family	Model	ARC-Easy no-norm	ARC-Easy norm	HellaSwag norm	HellaSwag no-norm	LAMBADA	PIQA norm	PIQA no-norm	BLiMP	MNLI
GPT-2	Small	44.4%	40.2%	31.6%	28.9%	32.4%	62.7%	62.9%	82.2%	34.0%
	Medium	48.6%	43.2%	39.9%	33.4%	41.9%	65.9%	66.4%	82.6%	35.3%
	Large	53.2%	46.0%	45.5%	36.6%	48.0%	69.2%	70.3%	83.6%	35.7%
	XL	56.4%	49.5%	51.3%	40.3%	51.3%	70.3%	71.2%	**83.4%** ↓	36.6%
Pythia	70M	31.4%	32.2%	26.8%	26.5%	4.9%	53.3%	54.3%	63.2%	35.4%
	160M	38.3%	38.1%	29.9%	28.4%	13.4%	59.7%	58.8%	70.0%	35.4%
	410M	52.6%	45.5%	40.5%	33.9%	47.8%	67.5%	67.5%	81.2%	**33.9%** ↓
	1B	57.4%	49.1%	47.0%	37.7%	54.6%	69.9%	71.1%	82.1%	34.9%
	2.8B	64.6%	58.0%	59.4%	45.5%	63.6%	73.6%	73.5%	83.0%	36.3%
Llama 2	7B	75.5%	73.9%	76.1%	57.0%	73.4%	78.5%	78.0%	83.0%	43.6%
	13B	78.7%	76.5%	79.7%	60.3%	76.5%	80.3%	79.5%	**82.8%** ↓	**43.0%** ↓
	70B	82.7%	79.5%	84.3%	65.2%	79.3%	82.5%	81.6%	**82.4%** ↓	44.5%
Llama 3.1	8B	82.1%	82.5%	79.4%	60.7%	74.9%	80.8%	79.1%	81.7%	52.5%
	70B	85.2%	**81.6%** ↓	85.7%	67.4%	79.4%	84.1%	82.6%	83.0%	**51.4%** ↓

Reference scores. One of the main results discussed in Section 4.3 regarding Gen. PPL and BPB is that even if we match token entropy closely to that of natural text, these metrics still fail to correctly recover the rankings of autoregressive models. We use a tolerance of 0.02, one tenth of the approximately 0.2 entropy gap associated with the Small/XL reversal reported by Acquaviva (2026), which we reproduce for four of the five scorers in Figure 13. Even at this tighter tolerance, every scorer reverses at least one adjacent GPT-2 comparison.

To assess how sensitive these reversals are to the particular samples being scored, we resample the 128 generated samples with replacement 5,000 times and recompute the score. Table 10 reports how often the resulting rankings reverse the expected order: the same reversals recur in 59–96% of resamples for Gen. PPL and 80–99.7% for BPB.

Similarity to held-out text. We compare all four AR families using 1,000 generated samples per model and the same 1,000 held-out FineWeb samples. We use each model's beginning-of-text token, sample without top-k or nucleus truncation, and tune temperature toward the same entropy target, 5.471 ± 0.02. The evaluated samples are generated with a separate random seed. We retain all samples and report their measured entropy in Table 11.

The MAUVE settings are the same as for DLMs in Appendix C.4. Seven of the fourteen AR banks meet the entropy tolerance. GPT-2 is the only fully matched family: MAUVE and MAUVE-star reverse Small/Medium and Large/XL, while both JS metrics recover all three adjacent orders. The matched Pythia 70M/160M pair is correctly ordered by all four scores (Table 12). The remaining pairs include at least one bank outside the tolerance, so their reversals may reflect more entropy differences.

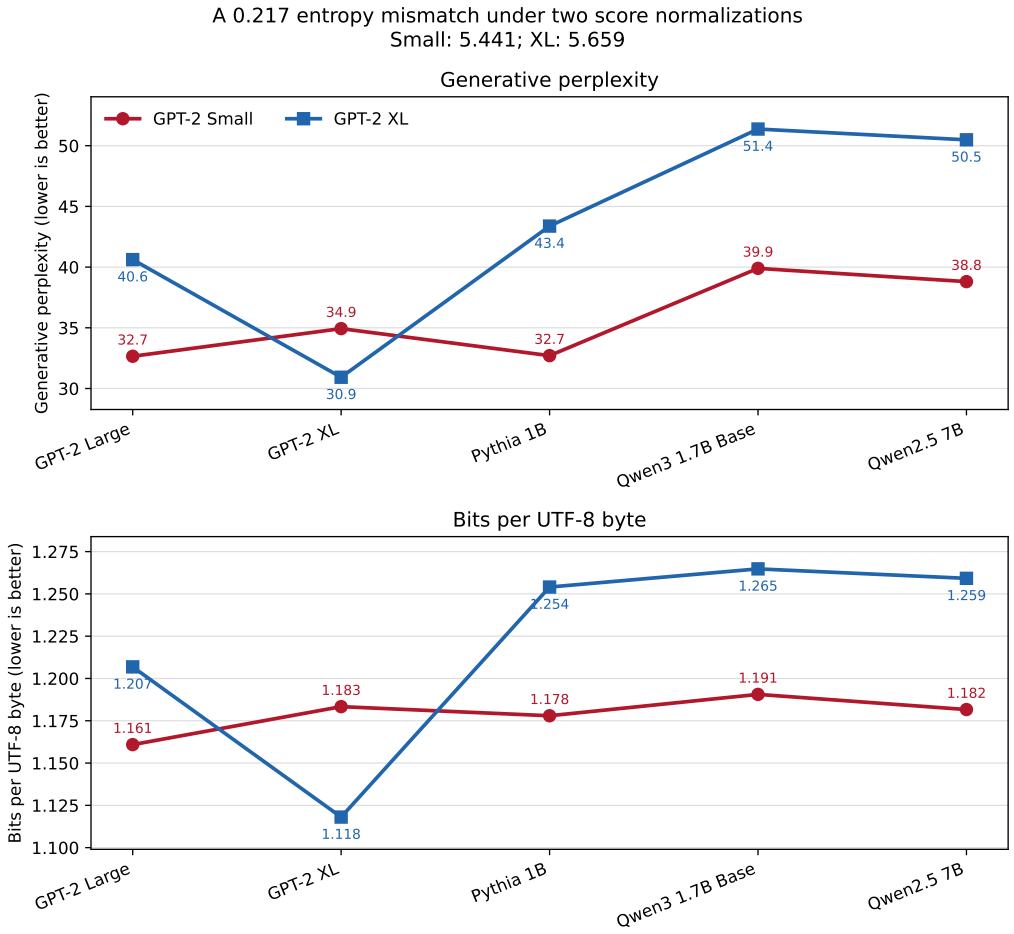


Figure 13: **An entropy mismatch alone reverses GPT-2 Small and XL.** With a 0.217 entropy mismatch, four of the five scorers shown (all except GPT-2 XL) prefer GPT-2 Small to GPT-2 XL under both Gen. PPL and BPB, reproducing Acquaviva (2026). This motivates our 0.02 tolerance. The reversals that remain at that tolerance are in Table 10.

Table 10: Autoregressive calibration after entropy matching. The FineWeb order is XL $>$ L $>$ M $>$ S, where $>$ means lower loss. Each observed order uses all 128 samples. The last column gives an adjacent reversed pair and the percentage of 5,000 resampled comparisons that remain reversed. Each comparison draws 128 samples with replacement independently from each bank and recomputes the score from total loss and total token or byte count.

Scorer	Unit	Observed GPT-2 order	Inverted pair; frequency
GPT-2 Large	Gen. PPL.	L $>$ XL $>$ M $>$ S	XL/L; 59.2%
GPT-2 XL	Gen. PPL.	XL $>$ M $>$ L $>$ S	L/M; 91.6%
GPT-Neo 2.7B	Gen. PPL.	XL $>$ M $>$ L $>$ S	L/M; 91.2%
Pythia 1B	Gen. PPL.	XL $>$ M $>$ L $>$ S	L/M; 95.7%
Qwen3 1.7B	Gen. PPL.	XL $>$ M $>$ L $>$ S	L/M; 95.3%
Qwen2.5 7B	Gen. PPL.	XL $>$ M $>$ L $>$ S	L/M; 95.9%
GPT-2 Large	BPB	L $>$ M $>$ XL $>$ S	XL/L; 80.5%
GPT-2 XL	BPB	XL $>$ M $>$ L $>$ S	L/M; 99.0%
GPT-Neo 2.7B	BPB	M $>$ XL $>$ L $>$ S	L/M; 99.2%
Pythia 1B	BPB	M $>$ XL $>$ L $>$ S	L/M; 99.7%
Qwen3 1.7B	BPB	M $>$ XL $>$ L $>$ S	L/M; 99.6%
Qwen2.5 7B	BPB	M $>$ XL $>$ L $>$ S	L/M; 99.6%

Table 11: **Distribution metrics on autoregressive samples.** Every model contributes 1,000 samples, compared with the same 1,000 held-out samples. MAUVE and MAUVE-star use GPT-2 Large representations and average three clustering seeds. JS uses word frequencies. T is the sampling temperature and H the measured average token entropy. A dagger marks an entropy outside 5.471 ± 0.02, determined before rounding. Larger MAUVE and smaller JS indicate closer agreement with held-out text. The sample count and MAUVE settings are the same as for DLMs (Appendix C.4).

Family	Model	T	H	MAUVE	MAUVE-star	Unigram JS	Bigram JS
GPT-2	Small	0.9180	5.480	0.745	0.797	0.137	0.571
	Medium	0.8856	5.478	0.663	0.721	0.136	0.568
	Large	0.9593	5.481	0.767	0.814	0.135	0.567
	XL	0.9622	5.486	0.752	0.807	0.133	0.564
Pythia	70M	0.9208	5.466	0.048	0.083	0.242	0.675
	160M	0.9284	5.472	0.373	0.456	0.165	0.596
	410M	1.0318	5.503†	0.136	0.190	0.199	0.667
	1B	1.0377	5.513†	0.236	0.304	0.187	0.654
	2.8B	1.0500	5.582†	0.140	0.195	0.189	0.666
Llama 2	7B	1.0170	5.490	0.602	0.671	0.152	0.602
	13B	1.0414	5.502†	0.410	0.489	0.176	0.633
	70B	1.0298	5.492†	0.731	0.787	0.144	0.595
Llama 3.1	8B	1.0542	5.437†	0.146	0.203	0.229	0.693
	70B	1.0472	5.339†	0.276	0.347	0.214	0.670

Table 12: **Does each distribution metric recover the FineWeb order?** Each arrow goes from the model with higher FineWeb loss to the one with lower loss. Red entries reverse that order. The final column says whether both generated banks fall inside the entropy window. A `no` means the comparison does not control entropy to the stated tolerance. These are comparisons of the observed averages, not significance tests or repeated temperature searches.

Family	FineWeb order	MAUVE	MAUVE-star	Unigram JS	Bigram JS	Both H matched?
GPT-2	Small→Medium	reversal	reversal	agrees	agrees	yes
	Medium→Large	agrees	agrees	agrees	agrees	yes
	Large→XL	reversal	reversal	agrees	agrees	yes
Pythia	70M→160M	agrees	agrees	agrees	agrees	yes
	160M→410M	reversal	reversal	reversal	reversal	no
	410M→1B	agrees	agrees	agrees	agrees	no
	1B→2.8B	reversal	reversal	reversal	reversal	no
Llama 2	7B→13B	reversal	reversal	reversal	reversal	no
	13B→70B	agrees	agrees	agrees	agrees	no
Llama 3.1	8B→70B	agrees	agrees	agrees	agrees	no

E.4 REDUNDANCY ACROSS FAMILIES

Two metrics are redundant for recipe selection when they repeatedly give the same seven-recipe order within the same training seed and shared checkpoint. As a complement, we provide in Figure 14 the Spearman correlation of those paired ranks and cluster the matrix.

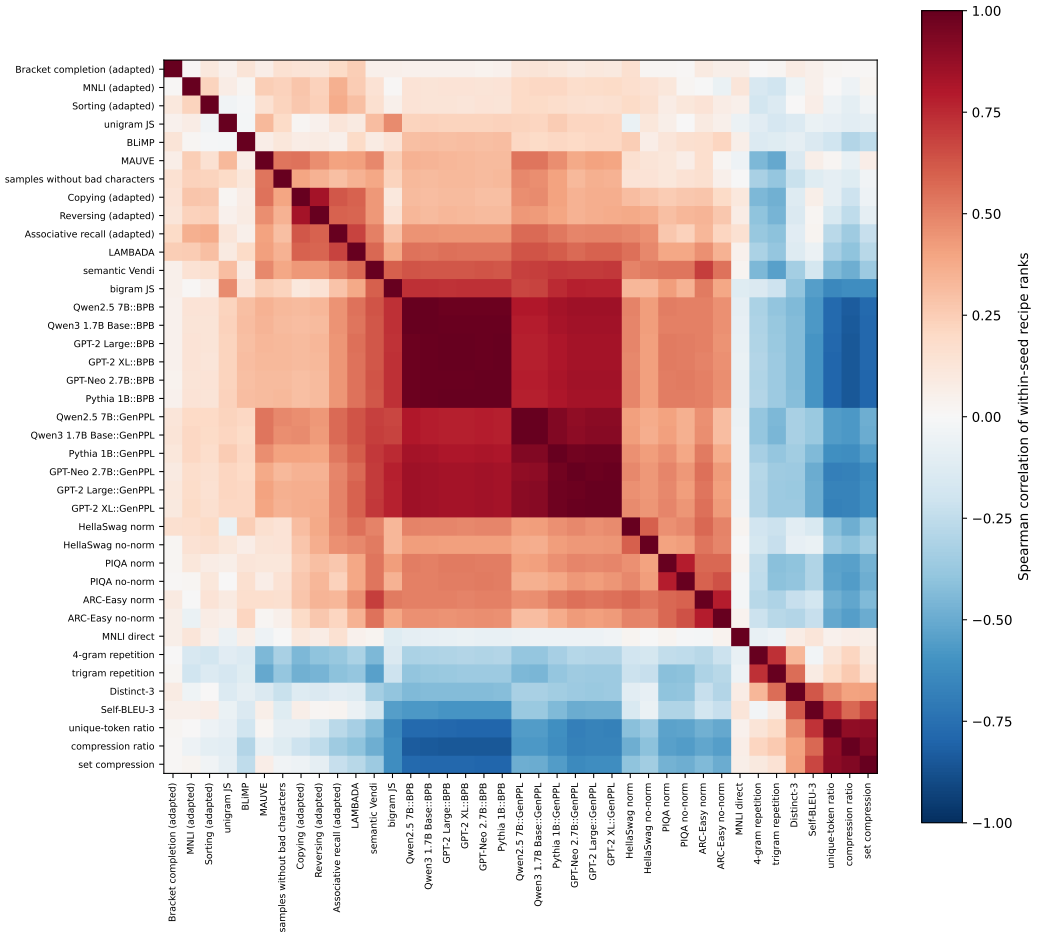


Figure 14: Spearman correlation between paired recipe ranks on the common ten-checkpoint grid. Reference-model scores agree closely with one another and generally move with HellaSwag, LAMBADA and several adapted tasks. Repetition and lexical-diversity measurements often move in the opposite direction. The clean-text check gives another view of the samples. Exact duplicates are omitted because every observed value is zero.

F HELLASWAG SPEEDRUN: PROTOCOL AND RESULTS

This appendix details the HellaSwag speedrun introduced in Section 5. We describe the benchmark setup and target, the procedure used to locate and verify finish times, the resulting training trajectories, and the variability associated with training seeds and evaluation corruptions.

F.1 BENCHMARK SETUP AND TARGET

The speedrun trains the seven reference recipes (Appendix B.1) with ten new independent training seeds per recipe, separate from the five seeds used in the metric audit. Checkpoints are evaluated on all 10,042 HellaSwag validation examples with the `norm` scoring convention selected in Section 5

and defined in Equation (1). A saved checkpoint c passes when

$$\overline{A}_{\text{HellaSwag}}(c) \geq 27.5\%, \tag{15}$$

where $\overline{A}_{\text{HellaSwag}}(c)$ is the mean accuracy of the ten runs at their saved checkpoints for the same elapsed minute. Each run saves one checkpoint per elapsed minute, at the end of the first training step that reaches that minute, and the reported finish time averages the ten recorded times of the selected checkpoint. Appendix F.2 describes how the passing checkpoint is located and verified.

The target is 2.5 percentage points above the 25% random-choice baseline, or about 251 additional correct answers on this validation split. While this might sound close to random chance, this is already enough to separate training recipes. Put in context with other similar scale projects, it also shows that this is fairly standard: the autoregressive nanoGPT speedrun (Jordan et al., 2024), which inspired our setup, stops at a validation-likelihood target at which models typically score slightly below 30% on HellaSwag. Note also that this target lies beyond the stage at which the recipes already produce mostly well-formed text: in the five-seed audit, 75% of the samples decode without malformed text after 5.2–23.4 minutes, when the average HellaSwag accuracy is still 25.2–26.8% (Figure 11). The audit trains every recipe for 3.3B tokens, whereas the speedrun runs continue until the target is reached, up to 9.83B tokens for D3PM uniform.

F.2 Locating and verifying the finish time

Scoring every saved checkpoint of every run would be expensive. We therefore locate the finish in four stages, estimating each answer score (Equation (1)) with 8, 32 or 128 random corruptions per candidate ending:

1. **Initial scan.** Evaluate minutes $1, 4, 7, \ldots, 28$, every ten minutes from minute 30, and the final checkpoint, using 32 corruptions per candidate ending.
2. **Interval refinement.** If this leaves a wide crossing interval $[l, r]$, evaluate with 8 corruptions the minutes $m_1 = \lceil 0.9r \rceil$, $m_{k+1} = \lceil 0.9\, m_k \rceil$, down to l.
3. **Minute-level search.** Evaluate every minute in the narrowed interval with 32 corruptions. The candidate is the first minute at which the ten-run mean reaches 27.5% and remains above it at all later measured minutes.
4. **Verification.** Reevaluate the candidate and its immediately preceding and following checkpoints with 128 corruptions: the preceding checkpoint must fail, and the selected and following checkpoints must pass. If these scores move the crossing, the search continues with 128 corruptions.

The selected checkpoint is thus the first passing minute with this verification and no reversal among later 128-corruption evaluations.

Note that across the 197 recipe–time points evaluated with both 8 and 32 corruptions (Figure 15), the two estimates differ by 0.27 percentage points on average and 1.09 points at most.

F.3 Training trajectories and benchmark results

Figure 3 in Section 5 gives the resulting ranking by finish time and its robustness to nearby targets. Figure 16 shows the underlying HellaSwag trajectories.

In this experiment, SUBS reaches the target first, after 12 minutes, and the masked recipes finish before the uniform-noise recipes. D3PM uniform does not reach the target within its observed run: after about 210 minutes and 9.83B tokens on eight GH200 GPUs, its ten-run mean is 27.088%. The trajectories show progress throughout training, whereas Table 13 documents the local crossing checks with 128 corruptions: for each finisher, the ten-run mean accuracy at the previous, selected and following minutes.

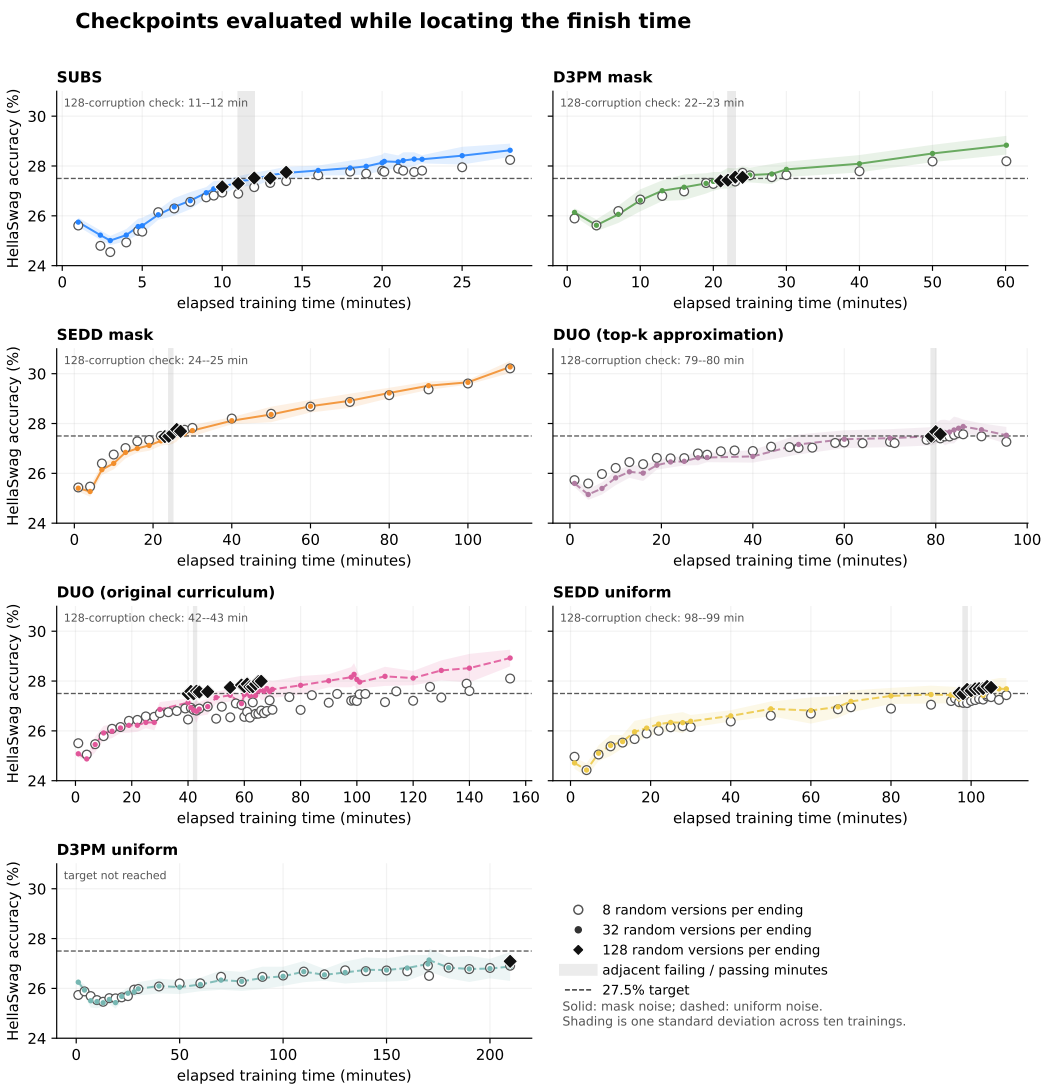


Figure 15: **Every checkpoint evaluated during the finish-time search.** Open rings use 8 corruptions per ending, coloured points 32, and black diamonds 128; each point averages ten trainings. Lines and shading show the 32-corruption average and one standard deviation across trainings; grey bands mark the failing and passing minutes around each selected checkpoint, verified with 128 corruptions (Table 13).

Table 13: Average HellaSwag accuracy (%) over ten trainings, using 128 corruptions per ending on all 10,042 examples. For each finisher, the previous minute is below 27.5% and the selected and following minutes are above it. D3PM uniform reaches 27.088% at 9.83B tokens, below the target.

Recipe	Minute	Previous	Selected	Following
SUBS	12	27.296	27.518	27.513
D3PM mask	23	27.433	27.553	27.545
SEDD mask	25	27.490	27.606	27.763
DUO (original curriculum)	43	27.488	27.594	27.560
DUO (top-k approximation)	80	27.493	27.674	27.572
SEDD uniform	99	27.488	27.665	27.610

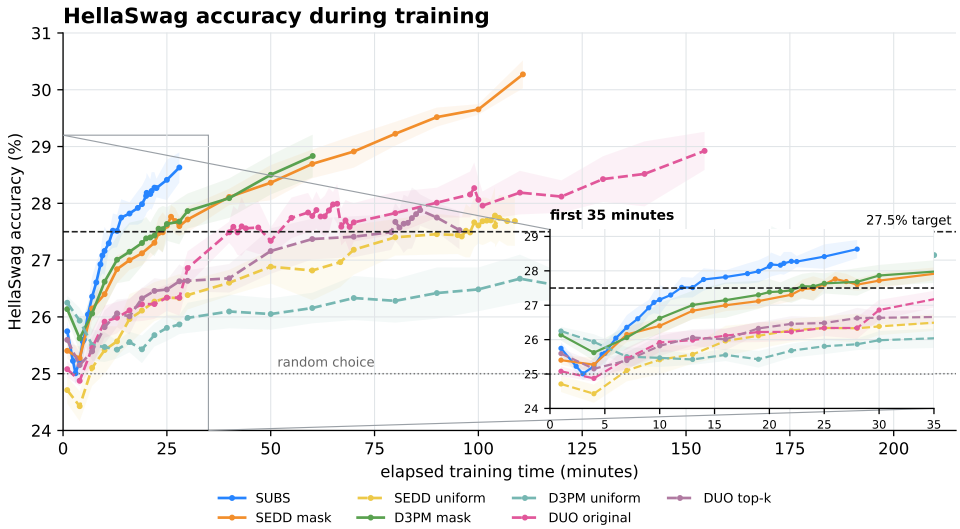


Figure 16: **HellaSwag accuracy against elapsed training time in the speedrun.** Lines average ten new trainings and shading is one standard deviation across trainings, dots mark evaluated checkpoints, and the inset enlarges the first 35 minutes.

F.4 Variability across training seeds and evaluation corruptions

Variation across training seeds. Independently trained models of the same recipe can differ because the training seeds change the model initialization and the training-time corruption noise (however, the data order is fixed for all recipes). If S training runs have mean HellaSwag accuracy $\bar{A}$ and standard deviation s_A, we report the Student-t 95% interval (assuming independent, approximately normal seed accuracies)

$$\bar{A} \pm t_{0.975,S-1}\frac{s_A}{\sqrt{S}}, \tag{16}$$

where $t_{0.975,S-1}$ is the 97.5th percentile of a Student-t distribution with $S-1$ degrees of freedom (2.78 for five seeds and 2.26 for ten). In the five-seed audit, the half-widths at the final HellaSwag checkpoint range from 0.14 to 0.62 percentage points across recipes. In the ten-seed speedrun, they range from 0.10 to 0.37 points at the selected checkpoints of the six recipes that reach the target (Table 14). The shading in Figures 15 and 16 shows one standard deviation across trainings, whereas Table 14 reports this 95% half-width. Passing uses the observed mean accuracy, so the interval can overlap 27.5% at the selected checkpoint.

Sensitivity to the number of evaluation corruptions. Each answer score in Equation (1) averages a finite number of random corruptions, so the same checkpoint can yield different accuracies. Equal corruption counts give every recipe the same evaluation budget, but not necessarily the same estimator variance: mask, uniform and DUO objectives define different losses and corruption processes. At a much larger scale, LLaDA uses 128 corruptions to stabilize the SUBS reconstruction-loss estimate (Nie et al., 2025, App. A.2); our speedrun uses 8, 32 and 128 corruptions as described in Appendix F.2. Table 14 measures the sensitivity to the corruption count at the selected checkpoint, or at the final checkpoint for D3PM uniform: for each training run, it compares the accuracy from the first 32 corruptions with the accuracy from all 128 corruptions of the same evaluation.

Table 14: Variation at the selected passing checkpoint, or at the final checkpoint for D3PM uniform. Each row covers ten trainings. Changes compare the first 32 with all 128 corruptions of the same evaluation. The last column is the 95% interval half-width across training seeds from Equation (16). Values are percentage points; observed changes are not bounds on estimation error.

Recipe	Minute	Average change	Largest change	Seed interval
SUBS	12	0.17	0.31	$\pm$0.22
D3PM mask	23	0.18	0.50	$\pm$0.26
SEDD mask	25	0.20	0.43	$\pm$0.10
DUO (original curriculum)	43	0.81	1.00	$\pm$0.37
DUO (top-k approximation)	80	0.27	0.70	$\pm$0.24
SEDD uniform	99	0.22	0.60	$\pm$0.31
D3PM uniform	end	0.22	0.54	$\pm$0.32