

REVISITING THE $16\times$ PRETRAINING COMPUTE GAP BETWEEN DIFFUSION AND AUTOREGRESSIVE LANGUAGE MODELS

Guanghan Wang^{1,†,‡} Bing He² Zhan Shi² Benoit Dumoulin²
 Hanqing Lu^{2,*} Volodymyr Kuleshov^{1,*}

¹Cornell University, Cornell Tech ²Amazon

Project page: <https://guanghanwang.com/dlm-scaling>

ABSTRACT

Diffusion language models (DLMs) have attracted wide attention for their fast, parallel decoding, but they are believed to be far less efficient to train than autoregressive language models (ARMs): a widely cited study reports that a DLM needs roughly $16\times$ the training compute of an ARM to match its likelihood, a figure that has made pretraining DLMs from scratch look prohibitive at scale. We find that the methodology behind this figure has fallen out of step with current DLM practice: it scores the model with its evidence lower bound (ELBO) and trains a full-sequence, decoder-only model. We therefore revisit the scaling laws of DLMs, adopting the exact likelihood under the model’s own decoding order, block diffusion training, an encoder–decoder architecture, and a sweep over the block sizes used in practice. Under this new protocol, we study across ten model variants, about 1,700 models in total, and find that pretraining DLMs is far cheaper than previously thought. Concretely, scoring with the exact likelihood reduces the compute multiple to $12.8\times$; block diffusion training and an encoder–decoder architecture further reduce it to $7.8\times$ at block size 32, and smaller block sizes narrow it to $4.6\times$ at block size 8 and $3.0\times$ at block size 4. We complement these scaling analyses with an empirical rule for the optimal decoder depth of encoder–decoder block diffusion, a held-out compute budget beyond the fitting range at which the gap between variants persists, and downstream benchmark results where likelihood improvements carry over to accuracy. Overall, the $16\times$ figure substantially overstates the pretraining cost gap between DLMs and ARMs, and pretraining DLMs from scratch is far more viable than previously assumed.

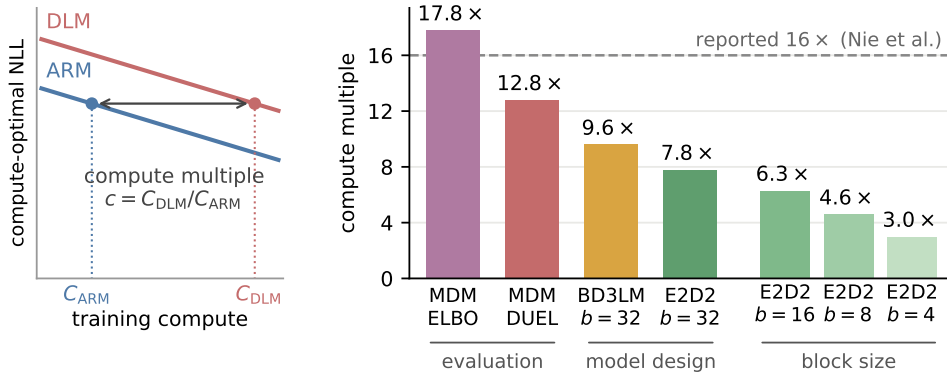


Figure 1: Revisiting the $16\times$ compute multiple between diffusion and autoregressive language models. Left: illustration of the compute multiple $c = C_{DLM}/C_{ARM}$ on the compute-optimal negative log-likelihood (NLL) scaling law plot. Right: c at 10^{20} FLOPs as the evaluation (ELBO $\rightarrow$ DUEL), the model design (MDM $\rightarrow$ BD3LM $\rightarrow$ E2D2) and the block size are revised.

[†]Work done during internship at Amazon.

^{*}Equal supervision.

[‡]Correspondence to: guanghan@cs.cornell.edu.

1 INTRODUCTION

Masked diffusion models (MDMs) have become the leading non-autoregressive alternative for language modeling (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2025; Nie et al., 2026). Their appeal is fast, parallel generation: open-weight masked diffusion language models report about $2.5\times$ the generation throughput of autoregressive language models (ARMs) at comparable accuracy (Wu et al., 2026a; Reda et al., 2026), with commercial systems reporting up to $10\times$ (Labs et al., 2025). Yet, their training efficiency is widely believed to be far worse than that of ARMs. The most cited quantification is the IsoFLOP study of Nie et al. (2025) (SMDM), which concludes that an MDM needs about $16\times$ the training FLOPs of an ARM to reach the same validation perplexity, and the number is now routinely quoted as a property of the MDM family. This $16\times$ figure has made pretraining MDMs from scratch look prohibitive at scale.

Taking a closer look at how this $16\times$ figure was achieved, however, we find that the methodology behind it has fallen out of step with current practice of diffusion language modeling. SMDM measured it on a full-sequence, decoder-only MDM scored with its negative evidence lower bound (ELBO), all three of which now have alternatives. First, the ELBO is an upper bound on the negative log-likelihood (NLL), and it scores the model under uniformly random token orders it never uses at inference; in contrast, DUEL (Turok et al., 2026) computes the exact NLL of a diffusion model under the custom decoding order it is deployed with. Second, nearly all of today’s large-scale diffusion language models (DLMs) use block-wise factorization (BD3LM; Arriola et al., 2025) with block sizes between 4 and 32, rather than operating over the full sequence. Third, encoder–decoder architectures that separate the processing of clean and noised tokens (E2D2; Arriola et al., 2026) offer computationally cheaper training than the decoder-only network. While the papers that introduce these techniques compare a handful of models at fixed sizes, none of these has been measured under a rigorous compute-matched IsoFLOP analysis.

We therefore revisit the scaling laws of masked diffusion models with a protocol that implements the ideas above: we score every diffusion model using the exact likelihood induced by its top-1 confidence decoding rule, utilize block-aware training and an encoder–decoder architecture, and sweep the block sizes $b \in \{4, 8, 16, 32\}$ which are commonly used in practice. In addition, we make two corrections to the SMDM protocol: we count training FLOPs exactly, including the output head and attention computation, and we read compute multiples within the measured range of the scaling laws rather than by extrapolation. Under this new protocol we study ten model variants and train about 1,700 models in total; to our knowledge, this is the largest compute-matched scaling analysis of diffusion language models to date.

With this analysis, we find that pretraining diffusion language models is far cheaper than previously thought (Figure 1). Scored with the ELBO, our full-sequence MDM lands close to the published figure, at $17.8\times$. Scoring the very same checkpoints with the exact likelihood lowers the multiple to $12.8\times$. Block diffusion training and the encoder–decoder architecture then reduce it further to $7.8\times$ at block size 32, and smaller block sizes narrow it to $4.6\times$ at block size 8 and $3.0\times$ at block size 4. In addition, we find that the optimal decoder depth of encoder–decoder block diffusion follows a simple rule that depends on the block size alone, that the gap between E2D2 and the ARM holds at a held-out budget three times larger than our largest fitted FLOPs, and that likelihood improvements carry over to downstream accuracy.

Our contributions are the following. (i) We revisit the scaling laws of diffusion language models under a revised protocol that reflects recent advances: exact likelihood evaluation, block diffusion, encoder–decoder models, and a sweep over practically used block sizes. (ii) We show that, under this protocol, pretraining diffusion language models is far cheaper than the $16\times$ suggests: exact likelihood evaluation reduces the compute multiple to $12.8\times$, block diffusion training with an encoder–decoder architecture reduces it to $7.8\times$ at $b=32$, and smaller block sizes further narrow it to $4.6\times$ at $b=8$ and $3.0\times$ at $b=4$. (iii) We complement these scaling analyses with an empirical rule for the optimal decoder depth of encoder–decoder block diffusion, which depends on the block size alone; a held-out budget at 3×10^{20} FLOPs, at which the gap between E2D2 and the ARM persists; and downstream benchmark results where likelihood improvements carry over to downstream accuracy.

2 BACKGROUND

Masked diffusion models. Let $\mathbf{x}^{1:n} = (x^1, \dots, x^n)$ be a sequence of n tokens from a vocabulary $\mathcal{V}$ augmented with a mask token m . The forward process of a masked diffusion model (Austin et al., 2021; Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2025) is indexed by $t \in [0, 1]$ and masks each token independently with probability $1 - \alpha_t$, where the schedule α_t decreases from $\alpha_0 = 1$ to $\alpha_1 = 0$; we write $\mathbf{z}_t^{1:n}$ for the partially masked sequence. A network $p_\theta(\cdot \mid \mathbf{z}_t^{1:n})$ predicts every masked position given the partially masked token sequence. Under the linear schedule $\alpha_t = 1 - t$, an MDM is trained with a negative ELBO,

$$\mathcal{L}_{\text{ELBO}}(\mathbf{x}^{1:n}) = \mathbb{E}_{t \sim \mathcal{U}(0,1)} \mathbb{E}_{\mathbf{z}_t^{1:n} \sim q_t(\cdot \mid \mathbf{x}^{1:n})} \left[\frac{1}{t} \sum_{\ell: z_t^\ell = m} -\log p_\theta(x^\ell \mid \mathbf{z}_t^{1:n}) \right] \geq -\log p_\theta(\mathbf{x}^{1:n}). \quad (1)$$

Following Sahoo et al. (2026), we train all diffusion models on the low-variance version of this loss that drops the weight $1/t$.

Prior scaling studies (Nie et al., 2025; Von Rütte et al., 2026; Ni et al., 2025b) evaluate a trained MDM by a Monte-Carlo estimate of Eq. 1 and compare it with the exact NLL of an ARM. This comparison is problematic in two respects. First, Eq. 1 is only an upper bound on the NLL; how far it lies from the NLL, and how this varies across models, is poorly understood. Second, Eq. 1 equals the expected NLL of an autoregressive factorization under a uniformly random token order (Hooeboom et al., 2021; Shi et al., 2024; Ou et al., 2025; Kim et al., 2025), whereas an MDM is never decoded in a random order in practice: its sampler chooses the order adaptively, most commonly by the model’s own confidence (Chang et al., 2022; Nie et al., 2026). Scoring a model under orders it never uses thus evaluates a distribution that differs from the one induced by its decoding rule at deployment.

Exact likelihood under a deterministic unmasking rule (DUEL). Turok et al. (2026) address both points. Let π be a deterministic unmasking rule that, given the tokens unmasked so far, selects the next position to reveal. Writing σ_i for the position revealed at step i , scoring $\mathbf{x}^{1:n}$ under π amounts to teacher-forcing the true tokens in the order σ , and $p_\theta^\pi(\mathbf{x}^{1:n}) = \prod_{i=1}^n p_\theta(x^{\sigma_i} \mid x^{\sigma_{<i}})$ is a properly normalized distribution, since σ_i depends only on $x^{\sigma_{<i}}$. Throughout the paper, we fix π to the top-1 confidence rule (Nie et al., 2026), the default sampler of diffusion language models, and report $-\log p_\theta^\pi(\mathbf{x}^{1:n})$ (DUEL) in nats per token like the ARM’s NLL.

Block diffusion. Full-sequence MDMs cannot generate sequences of arbitrary length and cannot reuse a KV cache. Block diffusion (BD3LM; Arriola et al., 2025) partitions $\mathbf{x}^{1:n}$ into contiguous blocks of b tokens, factorizes the sequence autoregressively over blocks, and models each block with a masked diffusion model conditioned on the clean prefix. Nearly all large-scale diffusion language models utilize block decomposition now. BD3LM keeps the decoder-only architecture: to score all blocks in one forward pass it concatenates the noised and clean sequences, so a training step runs over $2n$ tokens and costs roughly twice the FLOPs of an ARM or MDM step at the same size.

Encoder–decoder block diffusion. E2D2 (Arriola et al., 2026) avoids this overhead by splitting the L -layer transformer into an L_{enc} -layer encoder that represents the clean prefix and an $L_{\text{dec}} = L - L_{\text{enc}}$ -layer decoder that denoises the noisy block, attending to its own block by self-attention and to the encoder outputs of the prefix by cross-attention. Each training step is therefore cheaper than a BD3LM step because clean and noised tokens do not go through the same big network.

IsoFLOP scaling laws and the compute multiple. Prior scaling studies of diffusion language models (Nie et al., 2025; Von Rütte et al., 2026; Sahoo et al., 2026; Yang et al., 2026) adopt one IsoFLOP protocol from Hoffmann et al. (2022): for a fixed training budget C , a ladder of model sizes N is trained on $D = C/c(N)$ tokens each, where $c(N)$ is the training cost per token. Then, a parabola fitted to the loss against $\log N$ gives the compute-optimal loss $\mathcal{L}^*(C)$ at that budget. Across budgets they fit the power law $\mathcal{L}^*(C) = aC^\alpha$ by least squares in log–log space. The compute gap between two model families is measured at fixed loss: if family X reaches loss $\mathcal{L}$ at budget $C_X(\mathcal{L})$ and family Y at $C_Y(\mathcal{L})$, the compute multiple is $C_X(\mathcal{L})/C_Y(\mathcal{L})$. It is in this sense that Nie et al. (2025) report a multiple of 16 between MDM and ARM.

3 SCALING LAW PROTOCOL

In this section, we introduce our revised scaling law protocol for masked diffusion language models. It inherits the protocol of SMDM (Nie et al., 2025) with two kinds of changes. First, we replace three of SMDM’s choices with more recent alternatives: exact likelihood evaluation, block diffusion at commonly used block sizes, and an encoder–decoder architecture. Second, we correct two aspects of the original SMDM protocol, namely the FLOP count that sets the length of each run and the readout of the compute multiple from the fitted laws. Section 3.1 describes the first kind of changes, and Sections 3.2 and 3.3 state the resulting training and fitting procedure with both corrections included.

3.1 INCORPORATING RECENT ADVANCES

Exact likelihood evaluation. We score every diffusion model with DUEL under the top-1 confidence unmasking rule, yielding the exact likelihood induced by this decoding order. To keep the comparison with SMDM, we also score every diffusion model with the ELBO (Eq. 1, estimated with 128-particle Monte-Carlo), and the ARM with its exact NLL. All models are scored on one frozen validation set of 49,152 sequences, and unless stated otherwise the diffusion models are reported with DUEL.

Block diffusion. In our scaling law analysis protocol, we train block diffusion models at block sizes $b \in \{4, 8, 16, 32\}$. These configurations are chosen from the perspective of language model serving. By the roofline model (Williams et al., 2009), a GPU becomes compute-bound only above the ratio of its peak throughput to its memory bandwidth; taking BF16 serving as an example, this ratio is about 300 FLOPs per byte for an H100 GPU (NVIDIA, 2023). A block diffusion model decodes up to b tokens per sequence in one forward pass, which raises the arithmetic intensity to about Bb , where B is the batch size. For mid-size serving with B between 8 and 32, block sizes from 32 down to 8 thus bring Bb to about 256 and put the compute units left idle by autoregressive decoding to use, while a larger block adds little throughput. We therefore take $b \in \{8, 16, 32\}$ as the main block sizes, and use $b=4$ as a reference for how much training compute a still smaller block can save. These block sizes also match those used to train large diffusion language models: Nemotron-Labs-Diffusion (Fu et al., 2026) is trained with $b=32$, Nemotron-TwoTower (Reda et al., 2026) with $b=16$, and SDAR (Cheng et al., 2026) and FLARE (Zhu et al., 2026) with $b=4$ (see Table 4 in Appendix A.1 for more models).

Encoder–decoder architecture. Besides decoder-only block diffusion models (BD3LM), we also train encoder–decoder block diffusion models (E2D2) at the same four block sizes in our scaling law analysis protocol. Overall, this gives us ten model variants: ARM, MDM, BD3LM with $b \in \{4, 8, 16, 32\}$ and E2D2 with $b \in \{4, 8, 16, 32\}$.

Optimal decoder depth of E2D2. E2D2 has one degree of freedom that the other variants do not have: how the L layers of a given ladder size are split between the encoder and the decoder, i.e., the decoder fraction L_{dec}/L . A wrong split would penalize E2D2 for a reason unrelated to the architecture design itself, so at every grid point of E2D2, i.e., every combination of block size b , training budget C and model size N , we hold the total depth L fixed and sweep L_{dec} over a wide range. We then report the best DUEL value from the sweep. As a side product, we also analyze how the optimal decoder depth L_{dec}^* is related to b , C and N . We believe our results (Section 4.2) can give practical guidance on the pretraining of encoder–decoder block diffusion models.

3.2 TRAINING RECIPE AND ISOFLOP SWEEP

Training recipe. To keep our results consistent with the $16\times$ scaling analysis of SMDM, we largely inherit its choices of network architecture, data, tokenizer and hyperparameters. Concretely, all models are LLaMA-style transformers (Touvron et al., 2023) trained on SlimPajama (Sobolova et al., 2023) with a sequence length of 2048 and a batch of 256×2048 tokens. Following SMDM, we use standard parameterization (SP, as opposed to μP ; Yang et al., 2022) and AdamW ($\beta = (0.9, 0.95)$, weight decay 0.1, gradient clipping at 1.0) with a peak learning rate of 4×10^{-4} , a linear warm-up over 1% of the steps (at least 100) and cosine decay to 1/10 of the peak. All diffusion variants use the linear noise schedule $\alpha_t = 1 - t$ and, following Sahoo et al. (2026), the

low-variance training loss; following Arriola et al. (2026), BD3LM and E2D2 draw an independent noise level for every block and always mask at least one token in each block.

IsoFLOP grid. We reuse the size ladder of SMDM (19M–2.1B non-embedding parameters) for all variants and, for each variant, train 7–10 sizes around the compute-optimal size on each of five budgets $\{6 \times 10^{18}, 10^{19}, 3 \times 10^{19}, 6 \times 10^{19}, 10^{20}\}$ FLOPs. Across the ten variants, five budgets, sizes per budget and the decoder-depth sweep of E2D2, the grids comprise about 1,700 trained models (approximately 136k B200 GPU hours), more than any prior compute-matched study of diffusion language model scaling that we are aware of.

Compute accounting. At a FLOP budget C , the length of each run is set by the variant’s training FLOPs per token. SMDM takes this to be $6N$ (Kaplan et al., 2020), which is too coarse for a comparison across variants. The $6N$ rule ignores the attention score computation entirely, although at our sequence length of 2048 it accounts for a non-negligible fraction of the training FLOPs. More importantly, this fraction differs across variants. We therefore follow Hoffmann et al. (2022) and count the actual training FLOPs, including the output head and the score computation of every attention sublayer (Appendix B.1 gives the per-variant formulas).

3.3 FITTING SCALING LAWS

Fitting. For each variant we have five budgets C , each contributing 7–10 sizes N with their measured NLL (DUEL for the diffusion variants). For each (variant, C) we fit a centered parabola in $\log_{10} N$ to these values and take its vertex, which gives the compute-optimal NLL $\mathcal{L}^*(C)$; for E2D2, each size contributes only its best L_{dec} . Across budgets we fit the power law $\mathcal{L}^*(C) = aC^\alpha$ in log-log space, keeping the pure power-law form of prior work so that our numbers are comparable (see further discussion on the law’s functional form in Appendix C.4).

Reading the compute multiple. For each diffusion variant we take its compute-optimal NLL $\mathcal{L}^*$ at 10^{20} FLOPs, find the compute at which the ARM’s fitted law reaches that same $\mathcal{L}^*$, and take the ratio. Unlike SMDM, we do not extrapolate to do so: given the compute gap between the ARM and the diffusion variants, extrapolating a diffusion law to reach the ARM’s loss lands several times beyond the largest budget we ran, where a small level of noise in the fitted exponent would swing the multiple substantially. Reading the multiple inside the ARM’s measured range instead introduces little noise (see Appendix C.2 for a comparison between the two readouts).

4 RESULTS

4.1 THE $16\times$ OVERSTATES THE COST OF PRETRAINING DLMs

In Figure 2 we show the IsoFLOP curves of eight main model variants (those of the two $b=4$ variants are in Appendix C.1), and in Figure 3 the fitted scaling laws, whose parameters we list in Table 1. These laws fit the data well, with a maximum relative residual of only 0.5%. Next, we analyze one by one how each technique introduced in our revised protocol affects the scaling and the compute multiple of masked diffusion language models.

An ELBO baseline in line with $16\times$. With the ELBO readout, our MDM needs $17.8\times$ the ARM’s compute at 10^{20} FLOPs, comparable to the $16\times$ of Nie et al. (2025). Although our protocol differs from theirs (in the low-variance training loss, real FLOPs in place of $6ND$, and the readout), our compute multiple still agrees roughly with their $16\times$. On the same corpus and tokenizer, Sahoo et al. (2026) and Yang et al. (2026) report a smaller $\approx 14\times$ for the low-variance MDM, but with a DiT backbone (Peebles & Xie, 2023) in place of the Llama-style transformer of SMDM that we adopt. This baseline anchors the decomposition that follows.

Exact likelihood readout: the evaluation penalty. Scoring the very same MDM checkpoints with DUEL instead of the ELBO lowers the multiple to $12.8\times$. Nothing about the model or its training has changed; about a third of the reported gap is thus an artifact of the evaluation. A reduction is expected: the negative ELBO is an upper bound on the marginal NLL of the MDM

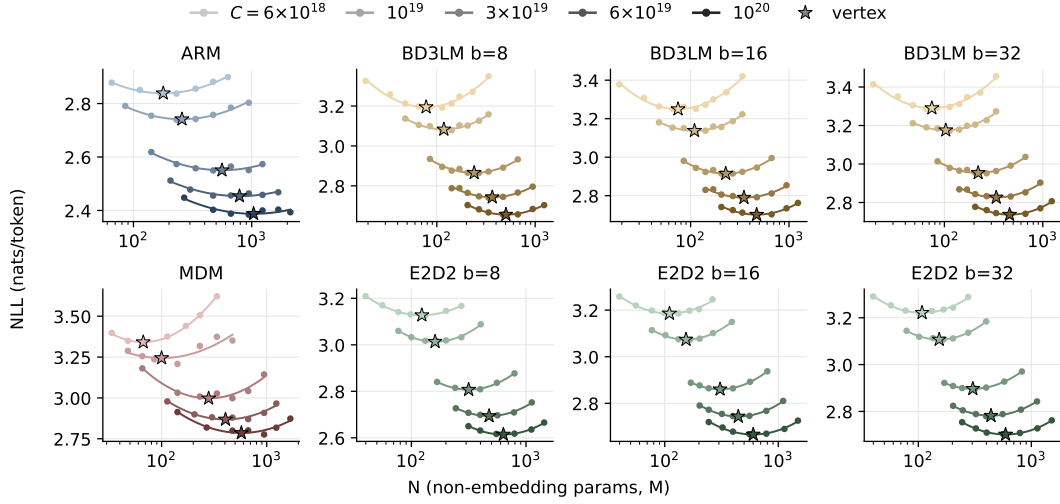


Figure 2: IsoFLOP curves for eight main model variants (DUEL for diffusion variants, exact NLL for the ARM; for E2D2, each point is the best L_{dec} at that size). Darker shades mark larger budgets, and stars mark fitted vertices.

Table 1: Fitted laws $\mathcal{L}^*(C) = a C^\alpha$ (DUEL for diffusion unless stated) and the compute multiple against ARM.

variant	a	α	multiple
ARM	40.95	-0.0618	—
MDM (ELBO)	49.93	-0.0623	17.8
MDM	57.87	-0.0659	12.8
BD3LM $b=32$	55.54	-0.0654	9.6
E2D2 $b=32$	47.79	-0.0624	7.8
BD3LM $b=16$	55.56	-0.0657	7.8
E2D2 $b=16$	49.26	-0.0634	6.3
BD3LM $b=8$	54.34	-0.0656	5.9
E2D2 $b=8$	48.52	-0.0635	4.6
BD3LM $b=4$	52.99	-0.0655	4.1
E2D2 $b=4$	45.50	-0.0627	3.0

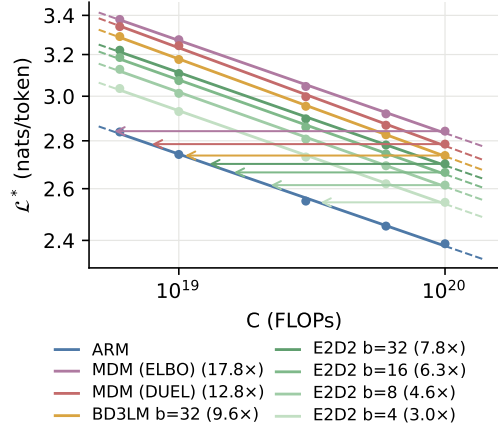


Figure 3: Compute-optimal NLL of different model variants, solid inside the fitted range and dashed outside.

Table 2: Compute multiples at 10^{20} FLOPs of every diffusion variant, measured with the negative ELBO and with DUEL. DUEL yields a smaller multiple than the negative ELBO for every variant.

	MDM	BD3LM 32	BD3LM 16	BD3LM 8	BD3LM 4	E2D2 32	E2D2 16	E2D2 8	E2D2 4
DUEL	12.8	9.6	7.8	5.9	4.1	7.8	6.3	4.6	3.0
ELBO	17.8	16.5	13.6	10.1	6.3	12.0	9.8	7.1	4.2

whose tightness is poorly understood, whereas DUEL is the exact NLL of the distribution induced by the model’s top-1 confidence decoding rule, and is therefore directly comparable with the ARM’s NLL. This is further supported by extra empirical evidence: in Table 2, where every diffusion variant is scored both ways, DUEL gives a smaller multiple than the negative ELBO in every case. Beyond likelihood, we also find that DUEL scores higher than the ELBO on multiple-choice benchmarks (Section 4.4), further evidence in favor of DUEL than the ELBO.

Block factorization: the first modeling change. The remaining 12.8 $\times$ is a genuine training-compute gap, and the next two replacements reduce it by changing how the model is built. Moving from the full-sequence MDM to BD3LM lowers the multiple to 9.6 $\times$ at $b=32$, 7.8 $\times$ at $b=16$ and 5.9 $\times$ at $b=8$. Although block-wise training has become common practice (Bie et al., 2025; Wu

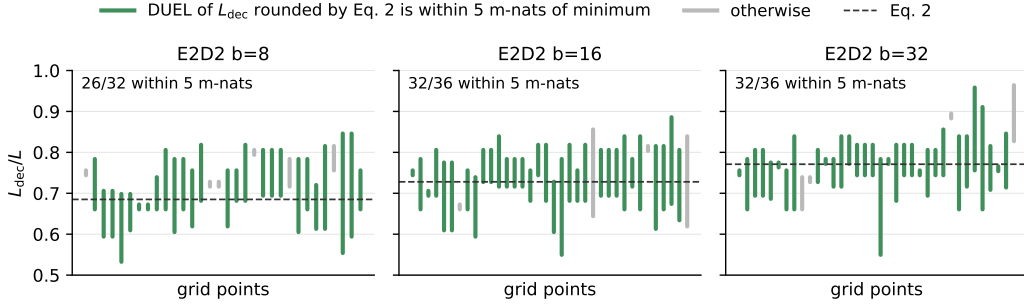


Figure 4: The decoder depth given by Eq. 2 is near-optimal for E2D2 at $b=8$, 16 and 32. Each bar represents one grid point (C, N) and spans the decoder fractions L_{dec}/L whose DUEL lies within 5 m-nats of the minimum of that grid point’s sweep. A bar is green if the DUEL of the decoder depth given by Eq. 2, i.e., its prediction multiplied by L and rounded to an integer, is within 5 m-nats of the minimum.

et al., 2026a; Fu et al., 2026; Reda et al., 2026; Cheng et al., 2026; Zhu et al., 2026), this is, to our knowledge, the first controlled, compute-matched demonstration of the benefit of this method in pretraining diffusion language models. We take this as evidence that introducing some degree of left-to-right bias helps language model learning, and that the likelihood gain from the left-to-right block factorization outweighs the penalty of BD3LM’s more expensive training steps, which at matched compute roughly halve the number of steps relative to the MDM.

Encoder–decoder architecture: the second modeling change. Moving from BD3LM to E2D2 at the same block size lowers the multiple once more, from $9.6\times$ to $7.8\times$ at $b=32$, from $7.8\times$ to $6.3\times$ at $b=16$ and from $5.9\times$ to $4.6\times$ at $b=8$; at $b=4$ it reaches $3.0\times$. Note that the decoder-only architecture, inherited from ARMs, remains the convention for diffusion language models: nearly all large-scale DLMs to date, including those trained with block diffusion, use a single decoder-only network (Nie et al., 2026; Ye et al., 2025; Wu et al., 2026a; Fu et al., 2026). Our experiments provide strong evidence that this convention is not the most training-efficient choice for masked diffusion: at matched compute, the encoder–decoder architecture is consistently better, which suggests that it is better suited to the masked diffusion training task. This shows that, for diffusion language models, a conceptual split between encoding and decoding is beneficial: although the encoder and the decoder each have less capacity than the original unified model, letting each concentrate on a different job allows them to use the pretraining compute more efficiently.

4.2 WHERE THE LAYERS GO: OPTIMAL DECODER DEPTH IN E2D2

The decoder-depth sweeps of Section 3.1 give the optimal decoder fraction L_{dec}^*/L at every one of the 142 (b, C, N) grid points of E2D2. They lead to a simple conclusion: **the optimal decoder fraction depends on the block size alone, and it is decoder-heavy**. A regression of L_{dec}^*/L on $\log_{10} N$, $\log_{10} C$ and $\log_2 b$ finds $\log_2 b$ to be the dominant predictor, with the dependence on N and C at most weak (Appendix C.3); regressing on $\log_2 b$ alone gives the empirical rule

$$L_{\text{dec}}^*/L \approx 0.556 + 0.043 \log_2 b. \quad (2)$$

The rule is convenient because the DUEL basin around L_{dec}^* is flat, so a slightly wrong choice costs little. To apply the rule, one multiplies it by the total depth L and rounds to a whole number of decoder layers. Figure 4 shows that this choice is nearly optimal: at 90 of the 104 grid points with $b \in \{8, 16, 32\}$, its DUEL lies within 5 m-nats of the best split found by the sweep, and within 10 m-nats at 99 of them, including every compute-optimal one (see Appendix C.3 for $b=4$).

4.3 THE GAP BETWEEN VARIANTS PERSISTS BEYOND THE FITTING RANGE

The laws of Section 4.1 are fitted on five budgets spanning 1.2 decades, and the natural question is whether the training efficiency gain they describe holds beyond that range. Given the cost of a full sweep at a larger budget, we proceeded as follows: at 3×10^{20} FLOPs we trained the ARM and E2D2

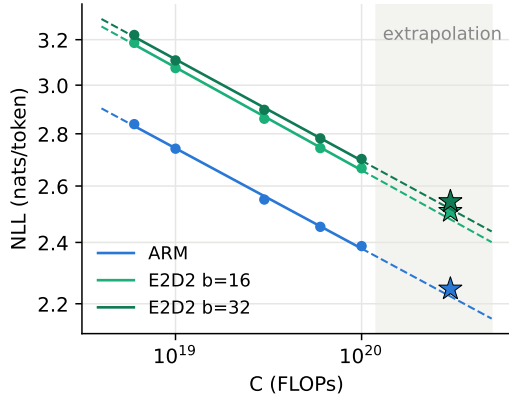


Figure 5: The held-out budget at 3×10^{20} FLOPs. Lines are the power laws fitted on the five budgets $\leq 10^{20}$ (dots), dashed outside the fitted range; stars are the measured values at the held-out budget.

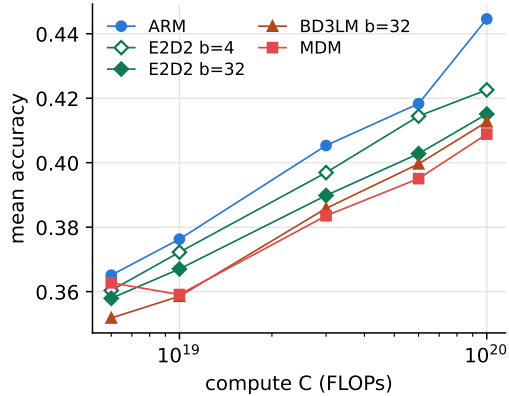


Figure 6: Mean accuracy over the eight retained benchmarks against compute, for the compute-optimal cell of the ARM and of four diffusion variants. The order in benchmark accuracy broadly matches the order in NLL.

at $b \in \{16, 32\}$. Each variant was trained at the three ladder sizes that bracket the optimal model size predicted by its law, and E2D2 additionally at three values of L_{dec} centered on the prediction of Eq. 2. We compare the measured NLL of each variant at the held-out budget with the prediction of the law fitted on the five smaller budgets (Figure 5).

The gap between variants is reproduced; the absolute laws are not exact. All three variants exhibit a common upward deviation of about 1.1% (24–28 m-nats) from the pure power-law extrapolation, suggesting a mild misspecification of the functional form, which Appendix C.4 traces to a small irreducible term that the five budgets cannot pin down (we therefore conservatively keep the pure power-law form of SMDM and leave fits with an irreducible term to future work). Importantly, the deviation is shared across variants, and the ARM–E2D2 gap is reproduced almost exactly: 261.6 m-nats measured against 257 predicted at $b=16$, and 296.8 against 294 at $b=32$. The quantity this paper is about, the gap between variants, is thus insensitive to the misspecification.

4.4 DOWNSTREAM PERFORMANCE BROADLY TRACKS LIKELIHOOD

We hereby test whether the improvements in likelihood generalize to downstream benchmarks. Concretely, we take the union of the benchmarks of Nie et al. (2025) and Prabhudesai et al. (2026), eleven in all, and score the compute-optimal cell of every variant at every budget zero-shot with lm-evaluation-harness (Gao et al., 2021). We rank the candidate answers by the exact NLL for the ARM, and by both DUEL and the 128-sample Monte-Carlo ELBO for the diffusion variants. Three benchmarks (WinoGrande, BoolQ and COPA) are excluded because, at these scales, the models’ performance on them is indistinguishable from chance or from noise (Appendix C.5). For each variant and budget we record the mean accuracy over the remaining eight—HellaSwag, ARC-easy, PIQA, SciQ, RACE, SIQA, OpenBookQA and LAMBADA (Table 15 reports every benchmark)—and find two things.

Table 3: Difference between DUEL and the ELBO in the eight-benchmark mean accuracy, as mean $\pm$ s.d. over the five budgets of each variant. DUEL outperforms the ELBO with statistical significance.

	$b=4$	$b=8$	$b=16$	$b=32$
MDM			$+0.4 \pm 0.1$	
BD3LM	$+1.0 \pm 0.4$	$+0.9 \pm 0.2$	$+0.9 \pm 0.2$	$+0.8 \pm 0.2$
E2D2	$+1.0 \pm 0.5$	$+1.1 \pm 0.4$	$+1.0 \pm 0.3$	$+1.0 \pm 0.2$

DUEL is the better scoring rule. As Table 3 shows, DUEL yields a statistically significantly higher accuracy than the ELBO for every diffusion variant. The exact readout is thus the better one, which also justifies fitting the scaling laws on DUEL rather than on the ELBO.

Likelihood improvements carry over to downstream accuracy. Figure 6 plots the eight-benchmark mean against compute for ARM, MDM, BD3LM ($b=32$), E2D2 ($b=32$), and E2D2 ($b=4$). Across variants, the order in accuracy tracks the order in NLL—the ARM, then E2D2 at

$b=4$, E2D2 at $b=32$, BD3LM at $b=32$ and the MDM. The two orders coincide exactly at the three largest budgets and differ only within noise at the two smallest. Likelihood improvements in pre-training thus carry over to downstream accuracy. Full results on each of the eight benchmarks are reported in Table 15 in Appendix C.5.

5 RELATED WORK

As diffusion language models have attracted attention for their fast, parallel inference, a number of studies have examined how they scale. Nie et al. (2025) were the first to study the scaling of diffusion language models: with IsoFLOP sweeps of MDMs and ARMs on SlimPajama with Llama-style transformers, they found that MDM needs $16\times$ the compute of the ARM to reach the same likelihood. Since then, a number of scaling studies of diffusion language models have followed. Sahoo et al. (2026) and Yang et al. (2026) extend the IsoFLOP methodology to other types of diffusion language model: Sahoo et al. (2026) find that uniform-state diffusion needs $23\times$ the compute of the ARM, and Yang et al. (2026) find that continuous diffusion needs $20\times$. Von Rütte et al. (2026) measure scaling laws for masked, uniform and hybrid-noise diffusion language models on Nemotron-CC with a CompleteP-parameterized transformer. Interestingly, although they criticize the pure power-law form of Nie et al. (2025) for implicitly assuming a zero irreducible loss, their own fits return an approximately zero irreducible term within their compute range, and they adopt a pure power law in the end. Ni et al. (2025b) refit a Chinchilla-form scaling law for the MDM. Beyond the compute-optimal setting, Prabhudesai et al. (2026) and Ni et al. (2025a) study the scaling of masked diffusion language models in the data-constrained regime and find that, under multi-epoch training, the MDM eventually overtakes the ARM. Our study differs from all of these. To our knowledge, ours is the first to substantially lower the $16\times$ figure, revealing that pretraining a diffusion language model from scratch is far cheaper than previously assumed.

6 CONCLUSION

In this paper we revisited the scaling laws of diffusion language models, whose widely cited $16\times$ compute gap to ARMs (Nie et al., 2025) was measured with a methodology that has fallen out of step with current practice. We revisited and substantially revised their scaling law protocol, proposing to score the model with its exact likelihood, utilize block-aware training and the encoder–decoder architecture, and sweep the commonly used block sizes. In total, we trained about 1,700 models across ten model variants, to our knowledge the largest diffusion language model scaling analysis to date, and found that pretraining diffusion language models is far cheaper than previously thought. Exact likelihood evaluation reduces the compute multiple to $12.8\times$; block diffusion training and an encoder–decoder architecture reduce it further to $7.8\times$ at block size 32; and smaller block sizes narrow it to $4.6\times$ at block size 8 and $3.0\times$ at block size 4. We complemented these scaling analyses with a simple empirical rule for the optimal decoder depth of encoder–decoder block diffusion models, a held-out budget at which the gap between variants persists, and downstream benchmarks where likelihood improvements carry over to accuracy. In conclusion, the $16\times$ figure substantially overstates the compute gap between diffusion and autoregressive language models, and pretraining diffusion language models from scratch is far more viable than previously assumed.

Limitations. We hereby list the limitations of this work. Owing to the cost of the grid, all experiments use a single fixed random seed. The held-out budget was not run for $b=8$. When measuring DUEL we use only the top-1 confidence sampler, the default sampler of diffusion language models. The exact likelihood under other unmasking rules may differ. All experiments share one learning-rate and batch-size configuration; although Porian et al. (2024) show that not tuning the hyperparameters at every scale can distort a scaling law, we keep the recipe of Nie et al. (2025) for consistency with the protocol under which the $16\times$ was measured. Likewise, we fit pure power laws without an irreducible term for consistency with Nie et al. (2025); the consistent miss at the held-out budget suggests that a small irreducible term is present (Appendix C.4), and we treat the fitted exponents as local to our compute range. It should be noted that, rather than the exact form of the scaling laws themselves, this paper is more concerned with the comparison between variants.

REPRODUCIBILITY STATEMENT

In Section 3, we state the training recipe, the FLOP accounting, the IsoFLOP grids and the readout and fitting procedure in full. Appendix B gives the FLOP formulas and the benchmark evaluation protocol. We also list the per-cell ledgers with the NLL of every model in Appendix E, so that the scaling law results can be regenerated without running any code.

AI USE STATEMENT

Generative AI tools were used to help develop the code to conduct experiments and to assist with writing, including drafting certain sections of this paper. The research ideas, the experimental design, and the interpretation of the results are the authors’ own. The authors take full responsibility for the content of the paper.

ACKNOWLEDGMENTS

We thank Gilad Turok for helpful discussions on DUEL, and Youngsuk Park for helpful discussions and feedback on the manuscript.

REFERENCES

- Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.
- Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference. *Advances in Neural Information Processing Systems*, 38:15612–15644, 2026.
- Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.
- Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
- Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, 2022.
- Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11305–11315. IEEE, 2022.
- Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, 2026.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

- 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.
- Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.
- Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, et al. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. *arXiv preprint arXiv:2607.05722*, 2026.
- Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.
- Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- Emiel Hooeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- Andrej Karpathy. nanoGPT. <https://github.com/karpathy/nanoGPT>, 2022.
- Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.
- Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations*, pp. 66–71, 2018.
- Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.
- Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pp. 785–794, 2017.
- Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. <https://github.com/facebookresearch/xformers>, 2022.
- Quentin Lhoest, Albert Villanova Del Moral, Yacine Jernite, Abhishek Thakur, Patrick Von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. Datasets: A community library for natural language processing. In *Proceedings of the 2021 conference on empirical methods in natural language processing: system demonstrations*, pp. 175–184, 2021.
- Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.
- Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

- Jinjie Ni, Qian Liu, Longxu Dou, Chao Du, Zili Wang, Hang Yan, Tianyu Pang, and Michael Qizhe Shieh. Diffusion language models are super data learners. *arXiv preprint arXiv:2511.03276*, 2025a.
- Jinjie Ni, Qian Liu, Chao Du, Longxu Dou, Hang Yan, Zili Wang, Tianyu Pang, and Michael Qizhe Shieh. Training optimal large diffusion language models. *arXiv preprint arXiv:2510.03280*, 2025b.
- Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. In *International Conference on Learning Representations*, volume 2025, pp. 82974–82997, 2025.
- 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. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.
- NVIDIA. NVIDIA H100 tensor core GPU datasheet. <https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306>, 2023.
- Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *International Conference on Learning Representations*, volume 2025, pp. 64972–65009, 2025.
- 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 *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.
- Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *Advances in Neural Information Processing Systems*, 37:100535–100570, 2024.
- Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion beats autoregressive in data-constrained settings. *Advances in Neural Information Processing Systems*, 38:10581–10606, 2026.
- Fitsum Reda, John Kamalu, Roger Waleffe, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-twotower: Diffusion language modeling with pretrained autoregressive context. *arXiv preprint arXiv:2606.26493*, 2026.
- Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pp. 90–95, 2011.
- Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.
- Subham Sekhar Sahoo, Jean-Marie Lemerrier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models. *arXiv preprint arXiv:2602.15014*, 2026.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

- Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Common-sense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.
- Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.
- Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. Slimpajama: A 627b token cleaned and deduplicated version of redpajama. <https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama>, 2023.
- The PyTorch Lightning team. Pytorch lightning. <https://github.com/Lightning-AI/pytorch-lightning>, 2019.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 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*, volume 2026, pp. 36638–36663, 2026.
- Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.
- Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. In *International Conference on Learning Representations*, volume 2026, pp. 128353–128370, 2026a.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In *International Conference on Learning Representations*, volume 2026, pp. 57027–57051, 2026b.
- Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.
- Zhihan Yang, Wei Guo, Shuibai Zhang, Subham Sekhar Sahoo, Yongxin Chen, Arash Vahdat, Morteza Mardani, and John Thickstun. Continuous diffusion scales competitively with discrete diffusion for language. *arXiv preprint arXiv:2605.18530*, 2026.
- 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*, 2025.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Yuchen Zhu, Jing Shi, Chongjian Ge, Hao Tan, Yiran Xu, Wanrong Zhu, Jason Kuen, Koustava Goswami, Rajiv Jain, Yongxin Chen, et al. Flare: Diffusion for hybrid language model. *arXiv preprint arXiv:2606.01774*, 2026.

CONTENTS

1	Introduction	2
2	Background	3
3	Scaling law protocol	4
3.1	Incorporating recent advances	4
3.2	Training recipe and IsoFLOP sweep	4
3.3	Fitting scaling laws	5
4	Results	5
4.1	The $16\times$ overstates the cost of pretraining DLMs	5
4.2	Where the layers go: optimal decoder depth in E2D2	7
4.3	The gap between variants persists beyond the fitting range	7
4.4	Downstream performance broadly tracks likelihood	8
5	Related work	9
6	Conclusion	9
A	Additional background	16
A.1	Existing large-scale diffusion language models	16
A.2	Attention patterns of the model variants	16
B	Additional experimental details	16
B.1	FLOP accounting	16
B.2	Scaling-law runs	17
B.3	Compute cost	18
B.4	Benchmark evaluation protocol	19
C	Additional experimental results	20
C.1	IsoFLOP curves	20
C.2	Compute multiple	20
C.3	Decoder depth	20
C.4	Held-out budget	22
C.5	Downstream benchmarks	23
D	Assets	24
E	Per-cell results of the IsoFLOP sweeps	24

A ADDITIONAL BACKGROUND

A.1 EXISTING LARGE-SCALE DIFFUSION LANGUAGE MODELS

Table 4 lists the block sizes adopted by recent large-scale masked diffusion language models.

Table 4: Block sizes adopted by large-scale masked diffusion language models. [†]Trained with full-sequence masking; the block size is the semi-autoregressive block length used at inference (for Dream, the setting chosen by Fast-dLLM (Wu et al., 2026b)).

Model	Size	Block size
LLaDA [†] (Nie et al., 2026)	8B	32
Dream [†] (Ye et al., 2025)	7B	32
LLaDA2.0 (Bie et al., 2025)	16B, 100B	32
Fast-dLLM v2 (Wu et al., 2026a)	1.5B, 7B	32
Nemotron-Labs-Diffusion (Fu et al., 2026)	3B, 8B, 14B	32
Nemotron-TwoTower (Reda et al., 2026)	30B-A3B	16
SDAR (Cheng et al., 2026)	1.7B, 4B, 8B, 30B-A3B	4 (4–64 released)
FLARE (Zhu et al., 2026)	2B, 4B, 9B	4

A.2 ATTENTION PATTERNS OF THE MODEL VARIANTS

Figure 7 illustrates the attention masks of the four model variants on a short sequence, which is what the (query, key) pair counts of Appendix B.1 are derived from.

- **ARM** uses a causal mask: each position attends to itself and to all earlier positions, giving $\frac{n(n+1)}{2}$ pairs per layer.
- **MDM** runs full bidirectional attention over the partially masked sequence: every position, masked or not, attends to every other position, giving n^2 pairs.
- **BD3LM** keeps the decoder-only architecture but runs a single tower over the length- $2n$ concatenation of the clean sequence $\mathbf{x}$ and the noised sequence $\mathbf{x}_t$: the clean half attends block-causally to itself, and each noised block attends to itself and to the clean tokens of all earlier blocks, so that a token can be denoised given its own block and an exact clean prefix; logits are scored on the noised half only.
- **E2D2** separates the two roles into an encoder and a decoder: the encoder runs block-causal self-attention over the clean sequence, and the decoder runs block-diagonal self-attention within each noised block together with cross-attention to the encoder outputs of the strictly preceding blocks.

B ADDITIONAL EXPERIMENTAL DETAILS

B.1 FLOP ACCOUNTING

We count real training FLOPs identically for every variant. Unlike $6ND$, our count includes the output head and the attention score computation. Let n be the sequence length, d the width, L the number of layers, V the vocabulary size, P the non-embedding parameter count, and b the block size; the number of (query, key) pairs per layer depends on the attention mask: causal $\frac{n(n+1)}{2}$, full n^2 , block-causal $\frac{n^2}{2} + \frac{nb}{2}$, block-diagonal nb , and block strict-prefix $\frac{n^2}{2} - \frac{nb}{2}$. For E2D2, the L layers split into L_{enc} encoder and L_{dec} decoder layers with non-embedding parameter counts P_{enc} and P_{dec} , the latter including the cross-attention projections. For each variant, the per-sample

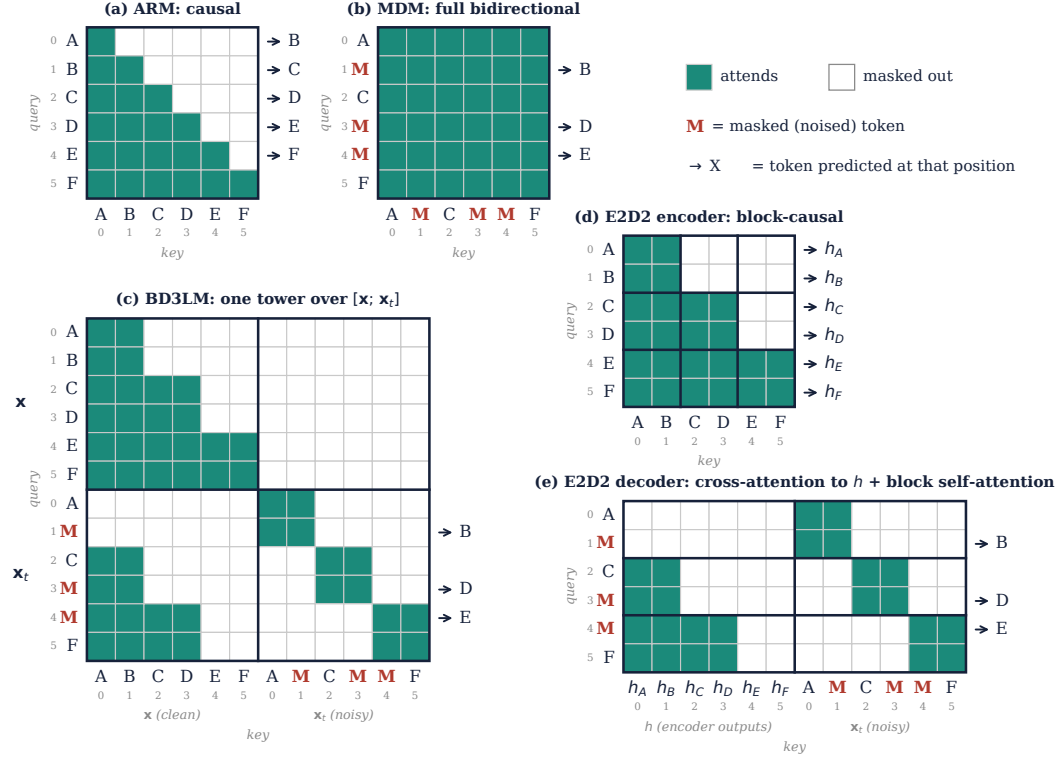


Figure 7: Attention masks of the four model variants on a six-token sequence with block size $b=2$. Rows are queries and columns are keys; filled cells attend. (a) ARM: causal self-attention. (b) MDM: full bidirectional self-attention over the partially masked sequence. (c) BD3LM: a single tower over the concatenation of the clean sequence $\mathbf{x}$ and the noised sequence $\mathbf{x}_t$; the clean half attends block-causally, and each noised block attends to itself and to the clean tokens of all earlier blocks. (d, e) E2D2: a block-causal encoder over the clean sequence, and a decoder with block-diagonal self-attention within each noised block plus cross-attention to the encoder outputs h of the strictly preceding blocks. **M** marks a masked (noised) token; arrows mark the token predicted at that position.

training FLOPs are computed as follows:

$$\text{ARM} = 6Pn + 6dLn(n+1) + 6dVn,$$

$$\text{MDM} = 6Pn + 12dLn^2 + 6dVn,$$

$$\text{BD3LM} = 12Pn + 12dL(n^2 + nb) + 6dVn,$$

$$\text{E2D2} = 6P_{\text{enc}}n + 6dL_{\text{enc}}(n^2 + nb) + 6P_{\text{dec}}n + 12dL_{\text{dec}}nb + 6dL_{\text{dec}}(n^2 - nb) + 6dVn.$$

Table 5 reports the attention score term as a fraction of the per-token training FLOPs for every variant across the size ladder. For BD3LM and E2D2 we take $b=32$ as the example, and for E2D2 we additionally set the decoder depth to half the total depth. Two patterns motivate counting real FLOPs rather than using $6ND$. First, the fraction is sizeable and grows toward smaller models, reaching about 30% for the MDM, so $6ND$ distorts the shape of $\mathcal{L}^*(C)$ along the ladder. Second, at matched size it differs markedly across variants: the MDM’s full bidirectional attention (n^2) costs about twice the ARM’s causal attention ($n^2/2$), while E2D2’s decoder attention is the cheapest. An attention-agnostic count therefore mis-charges the variants relative to one another, and this variant-specific difference, unlike terms common to all variants, does not cancel in the compute multiple.

B.2 SCALING-LAW RUNS

Tables 6 and 7 list every cell of the scaling-law grids: for each variant and budget, the ladder sizes N that were trained (non-embedding parameters in millions, as named in the SMDM ladder), and

Table 5: Attention score FLOPs as a percentage of per-token training FLOPs, by variant and model size ($b=32$; E2D2 decoder = $L/2$; sequence length 2048).

Size (M)	d	L	ARM	MDM	BD3LM	E2D2
19	384	8	16.8	28.8	20.3	16.0
34	512	8	14.4	25.1	16.9	13.6
66	640	10	13.2	23.4	14.9	12.4
113	768	12	12.0	21.5	13.2	11.2
170	768	18	12.7	22.6	13.6	11.8
268	1024	16	10.0	18.2	10.7	9.2
472	1280	18	8.4	15.5	8.9	7.7
761	1408	24	7.9	14.6	8.2	7.2
1233	1792	24	6.4	12.0	6.6	5.8
1678	2048	25	5.7	10.7	5.9	5.2
2121	2176	28	5.4	10.2	5.6	4.9

for E2D2 the decoder depths L_{dec} swept at each size. BD3LM at $b \in \{8, 16, 32\}$ share one grid; BD3LM at $b=4$ and E2D2 at $b=4$ use a slightly shifted one. The held-out budget of 3×10^{20} FLOPs (Section 4.3) is listed in its own rows.

Table 6: Ladder sizes N (millions of non-embedding parameters) trained at each budget for the ARM, the MDM and BD3LM.

variant	C (FLOPs)	trained sizes N
ARM	6×10^{18}	{66, 113, 170, 231, 336, 472, 629}
	10^{19}	{85, 142, 231, 336, 472, 666, 944}
	3×10^{19}	{142, 231, 336, 472, 666, 944, 1233}
	6×10^{19}	{206, 302, 472, 666, 944, 1233, 1678}
	10^{20}	{268, 472, 666, 944, 1233, 1678, 2121}
	3×10^{20} (held-out)	{1678, 2121, 2718}
MDM	6×10^{18}	{34, 48, 66, 113, 170, 231, 336}
	10^{19}	{48, 66, 85, 142, 231, 336, 472}
	3×10^{19}	{66, 142, 231, 336, 472, 666, 944}
	6×10^{19}	{113, 206, 302, 472, 666, 944, 1233}
	10^{20}	{142, 268, 472, 666, 944, 1233, 1678}
BD3LM $b \in \{8, 16, 32\}$	6×10^{18}	{19, 34, 48, 66, 85, 113, 142, 170, 231, 336}
	10^{19}	{48, 66, 85, 113, 142, 170, 206, 268, 336}
	3×10^{19}	{85, 113, 142, 170, 206, 268, 336, 472, 666}
	6×10^{19}	{142, 170, 206, 268, 336, 472, 666, 944}
	10^{20}	{206, 268, 336, 472, 666, 944, 1233}
BD3LM $b=4$	6×10^{18}	{34, 48, 66, 85, 113, 142, 170, 231, 336}
	10^{19}	{48, 66, 85, 113, 142, 170, 206, 268, 336}
	3×10^{19}	{142, 170, 206, 268, 336, 472, 666}
	6×10^{19}	{206, 268, 336, 472, 666, 944, 1233}
	10^{20}	{268, 336, 472, 666, 944, 1233, 1678}

B.3 COMPUTE COST

Table 8 lists the cost of the scaling law runs in B200 GPU hours, split into training and DUEL evaluation, for a total of approximately 136k GPU hours over 1,741 training runs (downstream benchmark evaluation not included). The E2D2 rows include the decoder-depth sweep, which is why they dominate the total. DUEL evaluation is by far the most expensive for the MDM, whose unmasking sweep spans the full sequence rather than one block at a time.

Table 7: E2D2 grid: for each budget and ladder size N (with total depth L), the range of decoder depths L_{dec} trained at each block size. “—” means the size was not trained at that block size.

C (FLOPs)	N (M)	L	$b=4$	$b=8$	$b=16$	$b=32$
6×10^{18}	34	8	2-7	2-7	2-7	2-7
	48	9	2-8	2-8	2-8	2-8
	66	10	2-9	2-9	2-9	3-9
	85	13	2-12	4-12	4-12	5-12
	113	12	2-11	4-11	3-11	3-11
	142	15	—	—	6-14	5-14
	170	18	2-17	8-16	8-17	8-17
	231	18	2-17	7-15	8-17	7-17
	336	20	2-19	—	—	—
	472	18	2-17	—	—	—
10^{19}	66	10	4-9	3-9	2-9	3-9
	85	13	4-12	5-12	4-11	6-12
	113	12	4-11	4-11	3-11	5-11
	142	15	3-14	6-14	6-14	8-14
	170	18	4-17	9-17	8-17	11-17
	231	18	4-17	8-16	6-16	10-16
	336	20	4-19	8-16	8-18	10-16
	472	18	4-17	—	—	—
3×10^{19}	142	15	4-14	8-14	6-14	8-14
	170	18	—	—	8-17	11-17
	206	16	4-15	8-15	7-15	9-15
	268	16	3-15	8-15	7-15	9-15
	336	20	4-19	10-18	9-19	12-18
	472	18	4-17	8-16	8-17	9-17
	666	21	4-20	11-19	10-20	11-20
	944	25	4-24	—	—	—
6×10^{19}	170	18	—	—	7-17	11-17
	206	16	4-14	8-15	7-15	9-15
	268	16	4-14	8-15	6-15	9-15
	336	20	4-19	10-18	10-19	13-19
	472	18	3-17	8-16	8-17	10-16
	666	21	4-20	10-18	10-20	12-20
	944	25	4-24	14-22	14-24	15-21
	1233	24	4-23	—	—	—
10^{20}	206	16	—	—	7-15	9-15
	268	16	5-14	8-15	7-15	9-15
	336	20	4-19	11-19	11-19	13-19
	472	18	4-17	8-15	8-17	11-17
	666	21	3-20	10-18	11-20	13-19
	944	25	4-24	14-22	14-24	16-22
	1233	24	4-23	12-20	14-23	14-23
	1678	25	4-24	—	—	—
3×10^{20} (held-out)	831	22	—	—	15-17	16-18
	944	25	—	—	16-20	18-20
	1233	24	—	—	16-18	17-21

B.4 BENCHMARK EVALUATION PROTOCOL

We obtain benchmark accuracies with lm-evaluation-harness (Gao et al., 2021). For HellaSwag (Zellers et al., 2019), ARC-easy (Clark et al., 2018), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), RACE (Lai et al., 2017) and OpenBookQA (Mihaylov et al., 2018) we report the `acc_norm` metric returned by the harness, and for SIQA (Sap et al., 2019), LAMBADA (Paperno et al., 2016), WinoGrande (Sakaguchi et al., 2021), BoolQ (Clark et al., 2019) and COPA (Roemmele et al., 2011) we report `acc`. The diffusion variants are scored with DUEL (top-1 confidence) and with the 128-sample ELBO. For LAMBADA the two readouts coincide, since the task is a single-word completion.

Table 8: Compute cost in B200 GPU hours by variant, split into training and DUEL evaluation. E2D2 rows include the decoder-depth sweep.

variant	runs	training	DUEL evaluation	total
ARM	43	1,507	—	1,507
MDM	80	2,393	8,384	10,777
BD3LM $b=32$	43	3,901	292	4,193
BD3LM $b=16$	43	3,903	147	4,050
BD3LM $b=8$	43	3,903	74	3,977
BD3LM $b=4$	40	3,504	39	3,543
E2D2 $b=32$	285	22,551	1,155	23,706
E2D2 $b=16$	355	28,082	718	28,800
E2D2 $b=8$	270	18,349	269	18,618
E2D2 $b=4$	539	36,960	327	37,287
total	1,741	125,053	11,405	136,458

C ADDITIONAL EXPERIMENTAL RESULTS

C.1 ISOFLP CURVES

Figure 8 shows the IsoFLOP curves of the two $b=4$ variants, which complement the eight variants in Figure 2.

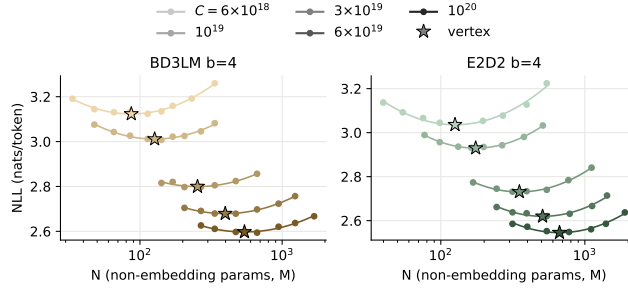


Figure 8: IsoFLOP curves for BD3LM and E2D2 at $b=4$ (DUEL; for E2D2, each point is the best L_{dec} at that size). Darker shades mark larger budgets, and stars mark fitted vertices.

C.2 COMPUTE MULTIPLE

Table 9 reports the compute multiple of every diffusion variant at every budget under the protocol of Section 3: the variant’s measured compute-optimal NLL at budget C is located on the ARM’s fitted law, and the multiple is C divided by the ARM compute that reaches the same NLL. Below 6×10^{19} FLOPs, the NLL of most diffusion variants lies above the ARM’s measured range (2.39–2.84 nats), so the ARM law has to be extrapolated below 6×10^{18} FLOPs; these entries are shown in parentheses for reference only. Table 10 reports the alternative, extrapolating readout of SMDM (Nie et al., 2025): the diffusion variant’s fitted law is extrapolated to the compute at which it reaches the ARM’s compute-optimal NLL at budget C , and the multiple is that compute divided by C . Since the fitted exponents of the diffusion variants are slightly steeper than the ARM’s, this readout decreases with C ; at 10^{20} FLOPs it differs from Table 9 by at most 22% (the MDM under DUEL), and the ordering of the variants is the same under both readouts.

C.3 DECODER DEPTH

Three readouts of the optimal split. Table 11 reads L_{dec}^*/L off the decoder-depth sweeps in three ways: the discrete argmin of DUEL, the vertex of a parabola fitted near the minimum (the readout used in the main text), and the argmin restricted to the compute-optimal N of each budget. All three agree closely at every block size, so the optimal split is insensitive to how it is read.

Table 9: Compute multiple against the ARM under the protocol of the main text (DUEL for diffusion unless stated), at each of the five budgets. Values in parentheses require extrapolating the ARM law below 6×10^{18} FLOPs and are shown for reference only; the last column is the value reported in Table 2.

variant	6×10^{18}	10^{19}	3×10^{19}	6×10^{19}	10^{20}
MDM (ELBO)	(17.5)	(17.6)	(16.3)	(16.5)	17.8
MDM	(14.6)	(15.1)	(12.6)	(12.4)	12.8
BD3LM $b=32$	(11.4)	(10.7)	(9.9)	9.7	9.6
BD3LM $b=16$	(9.3)	(8.8)	(8.0)	7.9	7.8
BD3LM $b=8$	(7.1)	(6.6)	(6.1)	6.0	5.9
BD3LM $b=4$	(4.9)	(4.5)	4.2	4.1	4.1
E2D2 $b=32$	(8.1)	(7.5)	(7.3)	7.5	7.8
E2D2 $b=16$	(6.7)	(6.3)	(5.9)	6.0	6.3
E2D2 $b=8$	(5.0)	(4.6)	4.4	4.5	4.6
E2D2 $b=4$	(3.1)	(2.9)	2.8	2.9	3.0

Table 10: Compute multiple against the ARM under the extrapolating readout of SMDM (DUEL for diffusion unless stated): the fitted law of the diffusion variant is extrapolated to the compute at which it reaches the ARM’s compute-optimal NLL at each budget.

variant	6×10^{18}	10^{19}	3×10^{19}	6×10^{19}	10^{20}
MDM (ELBO)	16.9	16.8	16.7	16.6	16.5
MDM	12.5	12.1	11.3	10.8	10.5
BD3LM $b=32$	9.8	9.5	8.9	8.6	8.4
BD3LM $b=16$	8.0	7.8	7.3	7.0	6.8
BD3LM $b=8$	6.2	6.0	5.6	5.4	5.3
BD3LM $b=4$	4.3	4.2	4.0	3.8	3.7
E2D2 $b=32$	7.6	7.6	7.5	7.4	7.4
E2D2 $b=16$	6.2	6.1	5.9	5.8	5.8
E2D2 $b=8$	4.6	4.5	4.4	4.3	4.3
E2D2 $b=4$	2.9	2.9	2.9	2.8	2.8

Table 11: Optimal decoder fraction L_{dec}^*/L of E2D2 at each block size, read off in three ways: the discrete argmin of DUEL over all columns, the vertex of a parabola fitted near the minimum over all columns, and the argmin restricted to the compute-optimal N of each budget. Entries are mean $\pm$ s.d.

b	4	8	16	32
argmin, all columns	0.645 ± 0.085	0.716 ± 0.040	0.728 ± 0.062	0.780 ± 0.055
vertex, all columns	0.643 ± 0.064	0.682 ± 0.029	0.723 ± 0.030	0.771 ± 0.049
at compute-optimal N	0.630 ± 0.045	0.706 ± 0.048	0.724 ± 0.034	0.756 ± 0.018

The smallest block size. Figure 9 repeats the analysis of Section 4.2 for $b=4$ (38 columns). The picture is the same as at the larger block sizes: the optimal split is decoder-heavy, and the three readouts agree (0.63–0.65, against 0.64 predicted by Eq. 2). Rounding Eq. 2 to the nearest integer number of decoder layers lands within 10 m-nats of the column minimum in 87% of columns and in all five compute-optimal columns (Table 12).

Regressions. We regress the parabola-vertex readout of L_{dec}^*/L over the 142 columns on $\log_{10} N$, $\log_{10} C$ and $\log_2 b$ jointly. The block size dominates: its coefficient is $+0.043$ per doubling of b ($t=12.7$). The model size has no detectable effect, with a coefficient of -0.007 per decade of N ($t=-0.5$). The budget has a small one, $+0.029$ per decade of C ($t=2.4$); across the 1.2 decades of our budgets this amounts to a shift of 0.035 in L_{dec}^*/L , less than one layer at every depth in the grid, whereas going from $b=4$ to $b=32$ shifts it by 0.13. We therefore regress on $\log_2 b$ alone, which gives Eq. 2. The rule describes the data well: it reproduces the mean vertex at each block size to within 0.005 (0.643, 0.682, 0.723 and 0.771 measured at $b=4, 8, 16$ and 32, against 0.642, 0.685, 0.728 and 0.771 predicted), and its residuals over the individual columns have an rms of 0.045 in L_{dec}^*/L , or 0.8 layers.

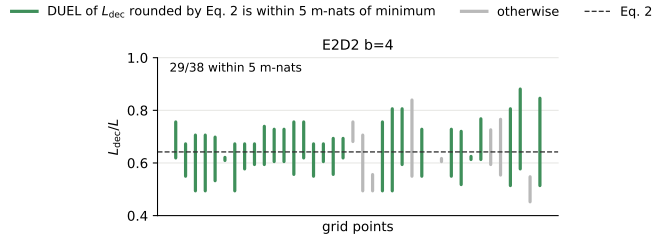


Figure 9: Same as Figure 4, for E2D2 at $b=4$.

Table 12: Rounded Eq. 2 against the column minimum, by block size.

b	columns	within 5 m-nats	within 10 m-nats	compute-optimal columns within 10 m-nats
4	38	76%	87%	5/5
8	32	81%	88%	5/5
16	36	89%	97%	5/5
32	36	89%	100%	5/5
all	142	84%	93%	20/20

C.4 HELD-OUT BUDGET

Table 13 compares, at the held-out budget of 3×10^{20} FLOPs, the NLL predicted by the power law fitted on the five budgets $\leq 10^{20}$ with the best measured cell of the ARM and of E2D2 at $b \in \{16, 32\}$. As reported in Section 4.3, the gap between E2D2 and the ARM is reproduced: 261.6 m-nats measured against 257 predicted at $b=16$, and 296.8 against 294 at $b=32$. At the same time, all three variants land 24–28 m-nats (1.07–1.15%) above their own law, a miss that is consistent in sign and size across the three.

Table 13: The held-out budget at 3×10^{20} FLOPs. Power laws fitted on the five budgets $\leq 10^{20}$ are compared with the best measured cell.

variant	law $\mathcal{L}^*(3 \times 10^{20})$	measured best cell	miss
ARM	2.2231	2.2468	+23.7 m-nats (+1.07%)
E2D2 $b=16$	2.4800	2.5084	+28.4 m-nats (+1.15%)
E2D2 $b=32$	2.5167	2.5436	+26.9 m-nats (+1.07%)

Fits with an irreducible term. A pure power law has no irreducible term, whereas the loss is bounded below by the entropy of the data (Hoffmann et al., 2022), a known weakness of the pure power law (Von Rütte et al., 2026; Ni et al., 2025b): a curve that bends towards a floor is under-predicted by a straight line in log-log space, by more the further one extrapolates, which is the signature observed. We therefore refit $\mathcal{L}^*(C) = E + a C^\alpha$ jointly over the eight DUEL variants (ARM, MDM, BD3LM and E2D2 at $b \in \{4, 16, 32\}$) with a single shared E and per-variant (a, α) , on the five budgets $\leq 10^{20}$, and profile the fit over fixed values of E (Table 14). Two observations follow. First, the in-sample rms decreases from 5.9 m-nats at $E = 0$ to 3.1 at $E \approx 1.5$, so the five budgets favor $E > 0$; but the minimum is shallow (3.1–3.9 m-nats for $E \in [1.2, 1.8]$), and the exponents move along a ridge with E , so the five budgets cannot determine the value of E . Second, adding $E \approx 1.5$ removes the miss at the held-out budget: at $E = 1.5$ the three 3×10^{20} cells are predicted to -7 , $+0.4$ and 0.0 m-nats (Figure 10). However, the held-out cells come from three sizes bracketing the predicted optimal model size rather than from a full IsoFLOP search, so we do not use them to fix E , and we conservatively keep the pure power law in the body of the paper, treating its exponents as local to the fitted range. We leave a scaling law with an irreducible term, which would require more budgets to pin down E , to future work.

Table 14: Profile of the shared- E fit on the five budgets $\leq 10^{20}$: in-sample rms, exponents, and the miss at the held-out budget (measured minus predicted, m-nats).

E	rms (m-nats)	α_{ARM}	$\alpha_{\text{E2D2 } 16}$	miss ARM	miss E2D2 16	miss E2D2 32
0.0	5.9	-0.062	-0.063	+23.7	+28.4	+26.9
1.0	4.1	-0.101	-0.097	+9.2	+14.2	+13.2
1.2	3.7	-0.115	-0.108	+4.1	+9.5	+8.7
1.4	3.2	-0.134	-0.122	-2.5	+3.8	+3.2
1.5	3.1	-0.147	-0.131	-6.7	+0.4	-0.0
1.6	3.2	-0.161	-0.141	-11.5	-3.5	-3.7
1.8	3.9	-0.203	-0.167	-24.1	-13.0	-12.7

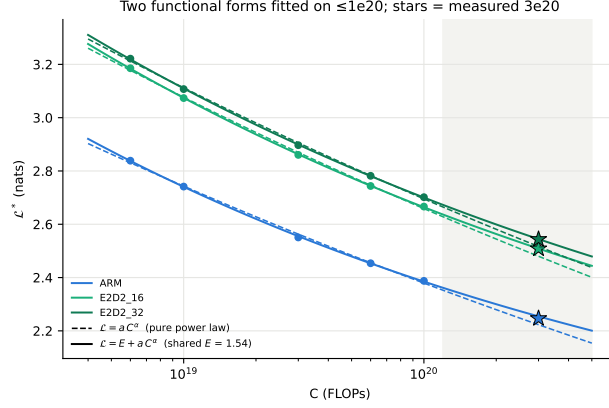


Figure 10: Pure power laws (dashed) and shared- E laws $\mathcal{L}^* = E + aC^\alpha$ (solid), both fitted on $\leq 10^{20}$, against the held-out budget at 3×10^{20} FLOPs (stars).

C.5 DOWNSTREAM BENCHMARKS

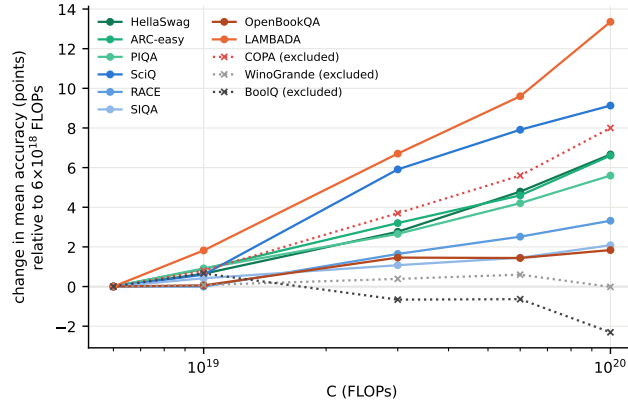


Figure 11: Benchmark screening. For each benchmark, the accuracy averaged over the ten variants at their compute-optimal cells, relative to its value at 6×10^{18} FLOPs. Dotted lines mark the three benchmarks we exclude.

Benchmark screening. We score the compute-optimal cell of every variant at every budget on all eleven benchmarks. Figure 11 shows, for each benchmark, how the accuracy averaged over the ten variants moves with compute. Eight benchmarks rise with compute by well more than their standard error. WinoGrande and BoolQ do not: on WinoGrande accuracy stays at chance at every budget (50.8% at 10^{20} FLOPs against a chance level of 50%), and on BoolQ it decreases with compute, so neither can distinguish a better model from a worse one at these scales. COPA does rise with compute, but it has only 100 questions, so the standard error of a single COPA accuracy is 4.8 points, against at most 2 points for every other benchmark; in a mean over nine benchmarks, COPA would account for about two-thirds of the variance. We therefore exclude these three benchmarks and use the remaining eight.

Per-benchmark accuracy. Table 15 lists the accuracy on each of the eight retained benchmarks, and their mean, for the compute-optimal cell of every variant at every budget; the means are the values plotted in Figure 6.

Table 15: Zero-shot accuracy (%) on the eight retained benchmarks at the compute-optimal cell of every variant at every budget (DUEL scoring for the diffusion variants; exact NLL for the ARM), and their mean. This is the data plotted in Figure 6.

variant	C	HellaSwag	ARC-e	PIQA	SciQ	RACE	SIQA	OBQA	LAMBADA	mean
ARM	6×10^{18}	28.9	35.1	61.1	60.1	27.0	37.0	26.2	16.6	36.5
	10^{19}	30.2	37.7	60.8	60.6	28.3	37.0	27.4	19.0	37.6
	3×10^{19}	34.1	40.4	63.2	65.9	29.8	37.4	27.2	26.3	40.5
	6×10^{19}	37.1	41.8	65.6	67.3	29.1	37.1	29.4	27.2	41.8
	10^{20}	40.1	42.9	67.7	72.4	33.3	38.3	30.2	30.8	44.5
MDM	6×10^{18}	28.5	32.1	56.2	63.1	27.0	36.7	25.4	21.1	36.3
	10^{19}	28.7	34.2	56.6	61.0	27.8	36.3	24.8	17.9	35.9
	3×10^{19}	30.5	36.3	58.8	64.7	30.1	37.3	25.0	24.2	38.4
	6×10^{19}	31.9	36.6	59.6	66.0	31.8	36.5	26.8	26.9	39.5
	10^{20}	33.4	39.2	61.0	68.5	32.0	36.7	26.8	29.6	40.9
BD3LM $b=32$	6×10^{18}	28.6	32.7	56.1	60.1	27.6	34.9	25.4	16.1	35.2
	10^{19}	29.3	33.4	56.8	62.0	26.8	36.0	25.2	17.5	35.9
	3×10^{19}	30.2	36.1	58.5	67.5	29.1	36.1	27.2	23.9	38.6
	6×10^{19}	32.3	37.2	60.2	70.0	29.9	38.0	26.4	25.8	40.0
	10^{20}	32.9	40.3	61.5	72.5	30.2	38.0	24.8	29.9	41.3
BD3LM $b=16$	6×10^{18}	28.3	33.5	55.8	60.0	27.8	35.6	23.6	15.5	35.0
	10^{19}	28.6	34.3	57.9	60.7	28.1	35.5	25.8	19.4	36.3
	3×10^{19}	30.6	36.2	58.9	67.5	27.8	36.2	28.0	24.3	38.7
	6×10^{19}	32.4	38.4	60.3	70.5	30.0	37.3	26.2	28.0	40.4
	10^{20}	34.3	40.2	61.6	71.2	30.2	37.0	27.4	30.8	41.6
BD3LM $b=8$	6×10^{18}	28.3	34.1	57.2	60.7	27.1	34.9	26.4	16.6	35.7
	10^{19}	29.0	34.4	58.3	61.4	28.3	36.4	24.6	18.6	36.4
	3×10^{19}	30.6	35.8	58.9	68.6	28.7	37.7	27.2	24.2	39.0
	6×10^{19}	32.8	37.2	60.9	70.3	30.5	37.0	25.2	26.3	40.1
	10^{20}	34.4	39.9	60.6	71.2	30.5	38.0	27.4	32.4	41.8
BD3LM $b=4$	6×10^{18}	28.4	32.7	55.7	61.1	28.3	36.2	27.0	17.4	35.9
	10^{19}	28.7	34.6	58.4	60.8	26.1	36.9	25.2	19.2	36.2
	3×10^{19}	31.1	36.4	59.7	64.9	29.7	37.1	25.8	23.2	38.5
	6×10^{19}	33.3	37.7	60.6	67.8	29.4	36.1	27.8	25.7	39.8
	10^{20}	35.4	40.7	63.4	66.8	30.6	37.8	27.8	31.5	41.8
E2D2 $b=32$	6×10^{18}	28.2	33.5	56.3	64.3	27.8	35.3	26.6	14.4	35.8
	10^{19}	28.7	34.6	58.1	65.6	27.8	35.2	26.2	17.5	36.7
	3×10^{19}	30.5	37.0	59.1	70.3	28.7	36.8	28.4	21.1	39.0
	6×10^{19}	32.3	37.7	60.5	72.4	31.4	36.6	26.8	24.6	40.3
	10^{20}	34.0	38.4	61.4	72.2	30.6	38.4	29.0	28.1	41.5
E2D2 $b=16$	6×10^{18}	28.6	33.7	56.6	64.0	28.0	34.8	26.0	17.1	36.1
	10^{19}	29.6	32.9	57.2	63.6	29.9	35.5	26.4	17.6	36.6
	3×10^{19}	31.2	36.3	58.9	68.2	29.8	36.1	26.8	20.8	38.5
	6×10^{19}	33.0	37.4	60.0	71.8	28.9	37.8	27.0	26.6	40.3
	10^{20}	34.9	39.5	61.1	71.0	30.7	37.6	27.2	28.5	41.3
E2D2 $b=8$	6×10^{18}	29.0	34.4	57.5	61.5	29.2	36.0	26.2	16.7	36.3
	10^{19}	29.6	33.7	56.9	62.2	26.4	36.4	27.4	20.4	36.6
	3×10^{19}	31.9	36.9	59.4	69.5	30.2	37.4	28.8	23.9	39.7
	6×10^{19}	33.4	38.9	61.9	69.0	30.8	38.0	28.8	25.9	40.8
	10^{20}	36.0	39.6	63.3	71.1	32.0	38.8	28.2	32.1	42.6
E2D2 $b=4$	6×10^{18}	28.9	34.9	57.1	60.2	28.2	36.2	24.6	18.1	36.0
	10^{19}	29.9	35.6	57.9	63.2	28.4	36.9	25.0	20.8	37.2
	3×10^{19}	32.8	37.3	60.9	67.1	30.5	36.5	27.6	24.8	39.7
	6×10^{19}	35.2	39.8	62.2	69.1	31.3	37.9	27.4	28.7	41.4
	10^{20}	37.1	41.9	64.0	69.5	30.9	38.1	27.0	29.6	42.3

D ASSETS

Table 16 lists the third-party code, data and benchmarks used in this study, with their licenses.

E PER-CELL RESULTS OF THE ISOFLP SWEEPS

Table 17 lists every cell of the ARM, MDM and BD3LM sweeps, with the size that minimizes each IsoFLOP curve in bold, and Tables 18–24 every trained model of the E2D2 decoder-depth sweeps, with the best decoder depth of each (budget, size) group in bold.

Table 16: Third-party assets used in this study and their licenses.

asset	license	asset	license
<i>Code</i>		<i>Data</i>	
SMDM (Nie et al., 2025)	Apache-2.0	SlimPajama-627B (Soboleva et al., 2023)	Apache-2.0
TinyLlama (Zhang et al., 2024) / lit-gpt	Apache-2.0	<i>Benchmarks</i>	
nanoGPT (Karpathy, 2022)	MIT	HellaSwag (Zellers et al., 2019)	MIT
GPT-NeoX (Black et al., 2022)	Apache-2.0	ARC-easy (Clark et al., 2018)	CC BY-SA 4.0
FlashAttention 2.6.3 (Dao, 2024)	BSD-3-Clause	PIQA (Bisk et al., 2020)	AFL-3.0
xFormers 0.0.28 (Lefaudeux et al., 2022)	BSD-3-Clause	SciQ (Welbl et al., 2017)	CC BY-NC 3.0
PyTorch 2.4.1 (Paszke et al., 2019)	BSD-3-Clause	RACE (Lai et al., 2017)	Apache-2.0
Lightning 2.1.2 (team, 2019)	Apache-2.0	SIQA (Sap et al., 2019)	CC BY 4.0
Transformers 4.31.0 (Wolf et al., 2020)	Apache-2.0	OpenBookQA (Mihaylov et al., 2018)	Apache-2.0
Datasets 3.0.1 (Lhoest et al., 2021)	Apache-2.0	LAMBADA (Paperno et al., 2016)	CC BY 4.0
SentencePiece (Kudo & Richardson, 2018)	Apache-2.0	WinoGrande (Sakaguchi et al., 2021)	Apache-2.0
lm-evaluation-harness 0.4.4 (Gao et al., 2021)	MIT	BoolQ (Clark et al., 2019)	CC BY-SA 3.0
		COPA (Roemmele et al., 2011)	BSD-2-Clause

Table 17: Per-cell results of the IsoFLOP sweeps for the ARM, the MDM and BD3LM. N is the nominal model size in millions of non-embedding parameters, as named in Table 6; entries are validation NLLs in nats per token (exact for the ARM; DUEL and ELBO for the diffusion variants), and “—” marks a size that was not trained for that variant. The last three ARM rows are the held-out budget of Section 4.3. Within each budget, the smallest value in each column, i.e. the minimum of each IsoFLOP curve, is in bold.

C	N	ARM	MDM		BD3LM $b=32$		BD3LM $b=16$		BD3LM $b=8$		BD3LM $b=4$	
		NLL	DUEL	ELBO	DUEL	ELBO	DUEL	ELBO	DUEL	ELBO	DUEL	ELBO
6×10^{18}	19	—	—	—	3.4143	3.5110	3.3783	3.4803	3.3263	3.4305	—	—
	34	—	3.3979	3.4465	3.3510	3.4422	3.3074	3.4069	3.2588	3.3601	3.1909	3.2756
	48	—	3.3491	3.4045	3.3057	3.4000	3.2640	3.3654	3.2125	3.3146	3.1442	3.2268
	66	2.8782	3.3355	3.3806	3.2863	3.3809	3.2504	3.3511	3.1987	3.2975	3.1310	3.2150
	85	—	—	—	3.2943	3.3836	3.2611	3.3577	3.1953	3.2954	3.1331	3.2176
	113	2.8507	3.3758	3.4006	3.2971	3.3883	3.2510	3.3505	3.1914	3.2895	3.1242	3.2081
	142	—	—	—	3.3115	3.4035	3.2722	3.3680	3.2130	3.3090	3.1347	3.2168
	170	2.8365	3.4406	3.4447	3.3480	3.4346	3.3004	3.3931	3.2480	3.3422	3.1593	3.2393
	231	2.8373	3.5058	3.4980	3.3709	3.4586	3.3369	3.4312	3.2711	3.3655	3.1918	3.2730
	336	2.8558	3.6221	3.6161	3.4562	3.5405	3.4207	3.5132	3.3495	3.4449	3.2598	3.3437
	472	2.8818	—	—	—	—	—	—	—	—	—	—
	629	2.8991	—	—	—	—	—	—	—	—	—	—
10^{19}	48	—	3.2884	3.3358	3.2118	3.3067	3.1806	3.2793	3.1372	3.2371	3.0758	3.1581
	66	—	3.2556	3.2969	3.1903	3.2867	3.1536	3.2548	3.1053	3.2049	3.0426	3.1240
	85	2.7910	3.2367	3.2701	3.1844	3.2755	3.1508	3.2476	3.0994	3.1950	3.0264	3.1066
	113	—	—	—	3.1728	3.2663	3.1323	3.2310	3.0750	3.1724	3.0121	3.0946
	142	2.7544	3.2083	3.2588	3.1787	3.2708	3.1379	3.2349	3.0788	3.1751	3.0078	3.0859
	170	—	—	—	3.1988	3.2860	3.1595	3.2539	3.1031	3.1968	3.0219	3.0986
	206	—	—	—	3.2044	3.2962	3.1567	3.2536	3.1020	3.1972	3.0253	3.1045
	231	2.7396	3.3178	3.3264	—	—	—	—	—	—	—	—
	268	—	—	—	3.2272	3.3165	3.1870	3.2812	3.1259	3.2186	3.0456	3.1233
	336	2.7420	3.3741	3.3765	3.2735	3.3602	3.2233	3.3187	3.1585	3.2521	3.0818	3.1587
	472	2.7577	3.3512	3.3951	—	—	—	—	—	—	—	—
	666	2.7845	—	—	—	—	—	—	—	—	—	—
	944	2.8030	—	—	—	—	—	—	—	—	—	—
3×10^{19}	66	—	3.1810	3.1794	—	—	—	—	—	—	—	—
	85	—	—	—	3.0139	3.1064	2.9810	3.0795	2.9349	3.0304	—	—
	113	—	—	—	2.9794	3.0741	2.9440	3.0435	2.8955	2.9913	—	—
	142	2.6185	3.0317	3.0796	2.9655	3.0577	2.9234	3.0198	2.8763	2.9709	2.8156	2.8916
	170	—	—	—	2.9693	3.0587	2.9333	3.0275	2.8848	2.9760	2.8203	2.8950
	206	—	—	—	2.9542	3.0467	2.9151	3.0108	2.8660	2.9594	2.7973	2.8708
	231	2.5735	3.0083	3.0509	—	—	—	—	—	—	—	—
	268	—	—	—	2.9516	3.0426	2.9151	3.0098	2.8646	2.9561	2.7974	2.8712
	336	2.5577	3.0269	3.0630	2.9635	3.0526	2.9240	3.0186	2.8716	2.9637	2.8053	2.8777
	472	2.5496	2.9985	3.0546	2.9915	3.0820	2.9457	3.0401	2.8910	2.9819	2.8240	2.8957
	666	2.5646	3.0424	3.0972	3.0372	3.1251	2.9956	3.0895	2.9326	3.0234	2.8565	2.9289
	944	2.5510	3.1436	3.1891	—	—	—	—	—	—	—	—
	1233	2.5734	—	—	—	—	—	—	—	—	—	—
6×10^{19}	113	—	2.9785	3.0238	—	—	—	—	—	—	—	—
	142	—	—	—	2.8683	2.9627	2.8299	2.9279	2.7880	2.8801	—	—
	170	—	—	—	2.8673	2.9582	2.8332	2.9271	2.7887	2.8803	—	—
	206	2.5119	2.8994	2.9542	2.8436	2.9350	2.8059	2.9016	2.7648	2.8552	2.7046	2.7763
	268	—	—	—	2.8330	2.9248	2.7982	2.8936	2.7475	2.8380	2.6911	2.7621
	302	2.4781	2.8813	2.9301	—	—	—	—	—	—	—	—
	336	—	—	—	2.8262	2.9188	2.7890	2.8847	2.7408	2.8310	2.6794	2.7497
	472	2.4557	2.8796	2.9259	2.8309	2.9243	2.7918	2.8875	2.7448	2.8338	2.6788	2.7480
	666	2.4560	2.8702	2.9216	2.8525	2.9433	2.8117	2.9058	2.7664	2.8553	2.6977	2.7665
	944	2.4637	2.9083	2.9622	2.9032	2.9912	2.8542	2.9474	2.7961	2.8862	2.7234	2.7926
	1233	2.4582	2.9653	3.0094	—	—	—	—	—	—	2.7568	2.8280
	1678	2.4680	—	—	—	—	—	—	—	—	—	—
10^{20}	142	—	2.9131	2.9637	—	—	—	—	—	—	—	—
	206	—	—	—	2.7775	2.8696	2.7412	2.8363	2.7037	2.7934	—	—
	268	2.4479	2.8189	2.8725	2.7607	2.8532	2.7252	2.8199	2.6823	2.7719	2.6262	2.6961
	336	—	—	—	2.7448	2.8363	2.7120	2.8063	2.6668	2.7557	2.6112	2.6803
	472	2.4025	2.7997	2.8529	2.7397	2.8323	2.7019	2.7963	2.6557	2.7438	2.5973	2.6647
	666	2.3881	2.8013	2.8565	2.7413	2.8335	2.7030	2.7979	2.6570	2.7451	2.5943	2.6619
	944	2.3843	2.7763	2.8401	2.7699	2.8604	2.7344	2.8268	2.6842	2.7719	2.6205	2.6879
	1233	2.4000	2.8177	2.8738	2.8066	2.8964	2.7625	2.8564	2.7034	2.7910	2.6372	2.7051
	1678	2.4031	2.8733	2.9289	—	—	—	—	—	—	2.6672	2.7348
	2121	2.3933	—	—	—	—	—	—	—	—	—	—
3×10^{20}	1678	2.2468	—	—	—	—	—	—	—	—	—	—
	2121	2.2610	—	—	—	—	—	—	—	—	—	—
	2718	2.2627	—	—	—	—	—	—	—	—	—	—

Table 18: Decoder-depth sweep of E2D2 $b=32$ (part 1 of 2): every trained model, one row per decoder depth. N is the nominal model size in millions of non-embedding parameters, as named in Table 6; DUEL and ELBO are validation NLLs in nats per token. L_{dec}/L is the decoder depth over the total depth. The best L_{dec} of each (budget, size) group is in bold. The rows at 3×10^{20} FLOPs are the held-out budget of Section 4.3.

N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO
$C = 6 \times 10^{18}$				$C = 6 \times 10^{18}$ (cont.)				$C = 3 \times 10^{19}$ (cont.)			
34	2/8	3.5005	3.5703	231	14/18	3.2890	3.3543	336	13/20	2.7882	2.8609
34	3/8	3.3989	3.4660	231	15/18	3.2972	3.3613	336	14/20	2.7859	2.8587
34	4/8	3.3500	3.4175	231	16/18	3.2975	3.3618	336	15/20	2.7795	2.8527
34	5/8	3.3105	3.3794	231	17/18	3.3110	3.3804	336	16/20	2.7807	2.8530
34	6/8	3.2909	3.3593	$C = 10^{19}$				336	17/20	2.7846	2.8568
34	7/8	3.2967	3.3690	66	3/10	3.2856	3.3547	336	18/20	2.7967	2.8690
48	2/9	3.4762	3.5476	66	4/10	3.2247	3.2932	336	19/20	2.8101	2.8832
48	3/9	3.3744	3.4421	66	5/10	3.1850	3.2533	472	10/18	2.8049	2.8772
48	4/9	3.3219	3.3888	66	6/10	3.1629	3.2330	472	11/18	2.7985	2.8719
48	5/9	3.2818	3.3501	66	7/10	3.1454	3.2150	472	12/18	2.7896	2.8639
48	6/9	3.2562	3.3238	66	8/10	3.1444	3.2144	472	13/18	2.7886	2.8609
48	7/9	3.2534	3.3226	66	9/10	3.1517	3.2256	472	14/18	2.7897	2.8630
48	8/9	3.2626	3.3335	85	6/13	3.1741	3.2438	472	15/18	2.7910	2.8647
66	3/10	3.3548	3.4228	85	7/13	3.1486	3.2199	472	16/18	2.7940	2.8677
66	4/10	3.3043	3.3710	85	8/13	3.1349	3.2035	666	12/21	2.8256	2.8977
66	5/10	3.2647	3.3304	85	9/13	3.1242	3.1940	666	13/21	2.8139	2.8865
66	6/10	3.2478	3.3148	85	10/13	3.1189	3.1893	666	14/21	2.8086	2.8815
66	7/10	3.2308	3.2984	85	11/13	3.1270	3.1962	666	15/21	2.8166	2.8901
66	8/10	3.2336	3.3023	85	12/13	3.1374	3.2090	666	16/21	2.8110	2.8839
66	9/10	3.2377	3.3105	113	5/12	3.1728	3.2425	666	17/21	2.8086	2.8807
85	5/13	3.2978	3.3643	113	6/12	3.1466	3.2149	666	18/21	2.8076	2.8803
85	6/13	3.2663	3.3340	113	7/12	3.1251	3.1934	666	19/21	2.8104	2.8829
85	7/13	3.2488	3.3172	113	8/12	3.1140	3.1831	666	20/21	2.8182	2.8915
85	8/13	3.2326	3.2989	113	9/12	3.1134	3.1822	944	15/25	2.8693	2.9421
85	9/13	3.2232	3.2913	113	10/12	3.1154	3.1861	944	16/25	2.8732	2.9439
85	10/13	3.2186	3.2869	113	11/12	3.1282	3.1998	944	17/25	2.8623	2.9334
85	11/13	3.2316	3.2988	142	8/15	3.1363	3.2061	944	18/25	2.8570	2.9289
85	12/13	3.2378	3.3084	142	9/15	3.1216	3.1917	944	19/25	2.8427	2.9156
113	3/12	3.3673	3.4342	142	10/15	3.1141	3.1830	944	20/25	2.8593	2.9326
113	4/12	3.3188	3.3869	142	11/15	3.1073	3.1768	944	21/25	2.8499	2.9225
113	5/12	3.2841	3.3501	142	12/15	3.1146	3.1825	$C = 10^{20}$			
113	6/12	3.2529	3.3204	142	13/15	3.1180	3.1874	206	9/16	2.7872	2.8599
113	7/12	3.2488	3.3149	170	14/15	3.1290	3.2015	206	10/16	2.7715	2.8436
113	8/12	3.2292	3.2974	170	11/18	3.1289	3.1980	206	11/16	2.7625	2.8349
113	9/12	3.2338	3.3021	170	12/18	3.1241	3.1910	206	12/16	2.7539	2.8269
113	10/12	3.2352	3.3019	170	13/18	3.1155	3.1845	206	13/16	2.7548	2.8273
113	11/12	3.2371	3.3085	170	14/18	3.1119	3.1797	206	14/16	2.7644	2.8369
142	5/15	3.3172	3.3820	170	15/18	3.1189	3.1871	206	15/16	2.7856	2.8591
142	6/15	3.2957	3.3613	170	16/18	3.1256	3.1951	268	9/16	2.7588	2.8322
142	7/15	3.2731	3.3385	170	17/18	3.1404	3.2114	268	10/16	2.7442	2.8158
142	8/15	3.2565	3.3223	231	10/18	3.1504	3.2163	268	11/16	2.7331	2.8063
142	9/15	3.2431	3.3104	231	11/18	3.1435	3.2112	268	12/16	2.7292	2.8018
142	10/15	3.2388	3.3050	231	12/18	3.1489	3.2176	268	13/16	2.7275	2.7995
142	11/15	3.2352	3.3007	231	13/18	3.1431	3.2116	268	14/16	2.7393	2.8122
142	12/15	3.2443	3.3089	231	14/18	3.1362	3.2052	268	15/16	2.7582	2.8322
142	13/15	3.2443	3.3105	231	15/18	3.1427	3.2099	336	13/20	2.7279	2.8002
142	14/15	3.2553	3.3250	231	16/18	3.1483	3.2149	336	14/20	2.7169	2.7898
170	8/18	3.2922	3.3582	336	10/20	3.2085	3.2751	336	15/20	2.7133	2.7856
170	9/18	3.2824	3.3490	336	11/20	3.1962	3.2628	336	16/20	2.7095	2.7810
170	10/18	3.2764	3.3404	336	12/20	3.1947	3.2617	336	17/20	2.7141	2.7866
170	11/18	3.2595	3.3261	336	13/20	3.1985	3.2643	336	18/20	2.7276	2.7993
170	12/18	3.2564	3.3208	336	14/20	3.1951	3.2607	336	19/20	2.7495	2.8237
170	13/18	3.2547	3.3190	336	15/20	3.1844	3.2501	472	11/18	2.7163	2.7891
170	14/18	3.2430	3.3068	336	16/20	3.1973	3.2634	472	12/18	2.7088	2.7827
170	15/18	3.2489	3.3154	$C = 3 \times 10^{19}$				472	13/18	2.7058	2.7787
170	16/18	3.2590	3.3253	142	8/15	2.9587	3.0279	472	14/18	2.7032	2.7761
170	17/18	3.2766	3.3464	142	9/15	2.9414	3.0128	472	15/18	2.7049	2.7789
231	7/18	3.3294	3.3938	142	10/15	2.9278	2.9984	472	16/18	2.7140	2.7871
231	8/18	3.3162	3.3822	142	11/15	2.9222	2.9945	472	17/18	2.7284	2.8024
231	9/18	3.3153	3.3790	142	12/15	2.9218	2.9919	666	13/21	2.7180	2.7915
231	10/18	3.2933	3.3573	142	13/15	2.9299	3.0020	666	14/21	2.7132	2.7874
231	11/18	3.2940	3.3569	142	14/15	2.9461	3.0189	666	15/21	2.7127	2.7861
231	12/18	3.2929	3.3579	170	11/18	2.9314	3.0031	666	16/21	2.7081	2.7805
231	13/18	3.2927	3.3571	170	12/18	2.9213	2.9923	666	17/21	2.7091	2.7829
				170	13/18	2.9097	2.9812	666	18/21	2.7144	2.7868
				170	14/18	2.9088	2.9813				

Table 19: Decoder-depth sweep of E2D2 $b=32$ (part 2 of 2): every trained model, one row per decoder depth.

N	L_{dec}/L	DUEL	ELBO
$C = 10^{20}$ (cont.)			
666	19/21	2.7170	2.7904
944	16/25	2.7424	2.8160
944	17/25	2.7359	2.8078
944	18/25	2.7294	2.8019
944	19/25	2.7291	2.8018
944	20/25	2.7277	2.8001
944	21/25	2.7293	2.8018
944	22/25	2.7353	2.8089
1233	14/24	2.7788	2.8515
1233	15/24	2.7746	2.8472
1233	16/24	2.7747	2.8472
1233	17/24	2.7796	2.8531
1233	18/24	2.7737	2.8456
1233	19/24	2.7682	2.8402
1233	20/24	2.7663	2.8392
1233	21/24	2.7632	2.8364
1233	22/24	2.7619	2.8352
1233	23/24	2.7655	2.8388
$C = 3 \times 10^{20}$			
831	16/22	2.5473	2.6205
831	17/22	2.5462	2.6192
831	18/22	2.5461	2.6193
944	18/25	2.5465	2.6188
944	19/25	2.5436	2.6166
944	20/25	2.5453	2.6182
1233	17/24	2.5548	2.6277
1233	18/24	2.5527	2.6258
1233	19/24	2.5543	2.6266
1233	20/24	2.5523	2.6261
1233	21/24	2.5562	2.6301

Table 20: Decoder-depth sweep of E2D2 $b=16$ (part 1 of 2): every trained model, one row per decoder depth. N is the nominal model size in millions of non-embedding parameters, as named in Table 6; DUEL and ELBO are validation NLLs in nats per token. L_{dec}/L is the decoder depth over the total depth. The best L_{dec} of each (budget, size) group is in bold. The rows at 3×10^{20} FLOPs are the held-out budget of Section 4.3.

N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO
$C = 6 \times 10^{18}$				$C = 6 \times 10^{18}$ (cont.)				$C = 10^{19}$ (cont.)				$C = 3 \times 10^{19}$ (cont.)			
34	2/8	3.4250	3.4991	231	14/18	3.2507	3.3210	336	13/20	3.1495	3.2191	666	10/21	2.9782	3.0506
34	3/8	3.3382	3.4073	231	15/18	3.2537	3.3226	336	14/20	3.1626	3.2336	666	11/21	2.9580	3.0307
34	4/8	3.2968	3.3658	231	16/18	3.2710	3.3424	336	15/20	3.1575	3.2273	666	12/21	2.9602	3.0322
34	5/8	3.2738	3.3441	231	17/18	3.2883	3.3614	336	16/20	3.1611	3.2298	666	13/21	2.9491	3.0217
34	6/8	3.2583	3.3294	$C = 10^{19}$				336	17/20	3.1530	3.2224	666	14/21	2.9481	3.0205
34	7/8	3.2755	3.3502	66	2/10	3.3111	3.3815	336	18/20	3.1663	3.2377	666	15/21	2.9588	3.0310
48	2/9	3.4005	3.4749	66	3/10	3.2211	3.2923	$C = 3 \times 10^{19}$				666	16/21	2.9437	3.0175
48	3/9	3.3136	3.3830	66	4/10	3.1700	3.2394	142	6/15	2.9565	3.0278	666	17/21	2.9373	3.0097
48	4/9	3.2617	3.3317	66	5/10	3.1397	3.2091	142	7/15	2.9341	3.0062	666	18/21	2.9458	3.0184
48	5/9	3.2391	3.3097	66	6/10	3.1244	3.1951	142	8/15	2.9180	2.9904	666	19/21	2.9460	3.0193
48	6/9	3.2199	3.2916	66	7/10	3.1138	3.1849	142	9/15	2.9029	2.9746	666	20/21	2.9511	3.0255
48	7/9	3.2239	3.2950	66	8/10	3.1156	3.1880	142	10/15	2.8941	2.9674	$C = 6 \times 10^{19}$			
48	8/9	3.2407	3.3146	66	9/10	3.1351	3.2110	142	11/15	2.8885	2.9623	170	7/18	2.8697	2.9412
66	2/10	3.3867	3.4558	85	4/13	3.1851	3.2561	142	12/15	2.8884	2.9622	170	8/18	2.8483	2.9210
66	3/10	3.2972	3.3679	85	5/13	3.1503	3.2202	142	13/15	2.9006	2.9754	170	9/18	2.8349	2.9086
66	4/10	3.2492	3.3184	85	6/13	3.1275	3.1989	142	14/15	2.9290	3.0048	170	10/18	2.8211	2.8938
66	5/10	3.2228	3.2934	85	7/13	3.1055	3.1782	170	8/18	2.9330	3.0043	170	11/18	2.8102	2.8828
66	6/10	3.2085	3.2788	85	8/13	3.0957	3.1664	170	9/18	2.9134	2.9856	170	12/18	2.8011	2.8745
66	7/10	3.1981	3.2685	85	9/13	3.0924	3.1648	170	10/18	2.9040	2.9768	170	13/18	2.7938	2.8681
66	8/10	3.2098	3.2800	85	10/13	3.0923	3.1647	170	11/18	2.8883	2.9611	170	14/18	2.7905	2.8646
66	9/10	3.2212	3.2965	85	11/13	3.0991	3.1707	170	12/18	2.8867	2.9590	170	15/18	2.7941	2.8689
85	4/13	3.2726	3.3412	113	3/12	3.2018	3.2715	170	13/18	2.8776	2.9512	170	16/18	2.8064	2.8813
85	5/13	3.2452	3.3136	113	4/12	3.1557	3.2266	170	14/18	2.8768	2.9506	170	17/18	2.8363	2.9135
85	6/13	3.2211	3.2902	113	5/12	3.1234	3.1945	170	15/18	2.8825	2.9559	206	7/16	2.8312	2.9036
85	7/13	3.2026	3.2727	113	6/12	3.1065	3.1757	170	16/18	2.8966	2.9718	206	8/16	2.8105	2.8830
85	8/13	3.1886	3.2575	113	7/12	3.0861	3.1570	170	17/18	2.9208	2.9975	206	9/16	2.7945	2.8673
85	9/13	3.1891	3.2597	113	8/12	3.0753	3.1475	206	7/16	2.9094	2.9820	206	10/16	2.7829	2.8565
85	10/13	3.1921	3.2617	113	9/12	3.0781	3.1499	206	8/16	2.8945	2.9677	206	11/16	2.7781	2.8517
85	11/13	3.2077	3.2783	113	10/12	3.0941	3.1679	206	9/16	2.8827	2.9558	206	12/16	2.7724	2.8463
85	12/13	3.2217	3.2959	113	11/12	3.1057	3.1820	206	10/16	2.8715	2.9454	206	13/16	2.7754	2.8493
113	3/12	3.3007	3.3700	142	6/15	3.1211	3.1919	206	11/16	2.8674	2.9412	206	14/16	2.7891	2.8648
113	4/12	3.2572	3.3283	142	7/15	3.1070	3.1779	206	12/16	2.8645	2.9384	206	15/16	2.8152	2.8917
113	5/12	3.2272	3.2967	142	8/15	3.0910	3.1630	206	13/16	2.8704	2.9444	268	6/16	2.8319	2.9042
113	6/12	3.2065	3.2759	142	9/15	3.0824	3.1528	206	14/16	2.8857	2.9607	268	7/16	2.8050	2.8790
113	7/12	3.1931	3.2626	142	10/15	3.0790	3.1496	206	15/16	2.9085	2.9850	268	8/16	2.7890	2.8623
113	8/12	3.1857	3.2549	142	11/15	3.0739	3.1461	268	7/16	2.9018	2.9755	268	9/16	2.7732	2.8471
113	9/12	3.1949	3.2657	142	12/15	3.0783	3.1502	268	8/16	2.8880	2.9610	268	10/16	2.7640	2.8378
113	10/12	3.2068	3.2778	142	13/15	3.0993	3.1723	268	9/16	2.8785	2.9525	268	11/16	2.7574	2.8315
113	11/12	3.2170	3.2914	142	14/15	3.1125	3.1881	268	10/16	2.8684	2.9425	268	12/16	2.7546	2.8283
142	6/15	3.2401	3.3078	170	8/18	3.1151	3.1865	268	11/16	2.8636	2.9379	268	13/16	2.7584	2.8329
142	7/15	3.2216	3.2906	170	9/18	3.1040	3.1738	268	12/16	2.8609	2.9362	268	14/16	2.7743	2.8496
142	8/15	3.2104	3.2791	170	10/18	3.0968	3.1679	268	13/16	2.8694	2.9436	268	15/16	2.7980	2.8746
142	9/15	3.2002	3.2702	170	11/18	3.0885	3.1608	268	14/16	2.8816	2.9567	336	10/20	2.7782	2.8520
142	10/15	3.1976	3.2686	170	12/18	3.0829	3.1545	268	15/16	2.9009	2.9774	336	11/20	2.7665	2.8401
142	11/15	3.1982	3.2673	170	13/18	3.0791	3.1495	336	9/20	2.9007	2.9746	336	12/20	2.7625	2.8362
142	12/15	3.2105	3.2819	170	14/18	3.0777	3.1490	336	10/20	2.8880	2.9624	336	13/20	2.7531	2.8276
142	13/15	3.2207	3.2915	170	15/18	3.0858	3.1583	336	11/20	2.8849	2.9588	336	14/20	2.7484	2.8216
142	14/15	3.2385	3.3136	170	16/18	3.1018	3.1741	336	12/20	2.8794	2.9528	336	15/20	2.7454	2.8200
170	8/18	3.2417	3.3105	170	17/18	3.1267	3.2009	336	13/20	2.8714	2.9457	336	16/20	2.7476	2.8215
170	9/18	3.2294	3.2986	231	6/18	3.1535	3.2230	336	14/20	2.8682	2.9424	336	17/20	2.7525	2.8261
170	10/18	3.2244	3.2947	231	7/18	3.1344	3.2038	336	15/20	2.8642	2.9382	336	18/20	2.7723	2.8472
170	11/18	3.2176	3.2878	231	8/18	3.1261	3.1975	336	16/20	2.8649	2.9396	336	19/20	2.7957	2.8720
170	12/18	3.2104	3.2787	231	9/18	3.1167	3.1863	336	17/20	2.8812	2.9559	472	8/18	2.7802	2.8549
170	13/18	3.2039	3.2739	231	10/18	3.1238	3.1949	336	18/20	2.8946	2.9697	472	9/18	2.7722	2.8477
170	14/18	3.2087	3.2780	231	11/18	3.1016	3.1733	336	19/20	2.9115	2.9881	472	10/18	2.7620	2.8357
170	15/18	3.2196	3.2914	231	12/18	3.1036	3.1749	472	8/18	2.9181	2.9915	472	11/18	2.7540	2.8289
170	16/18	3.2356	3.3060	231	13/18	3.1055	3.1774	472	9/18	2.9081	2.9829	472	12/18	2.7515	2.8256
170	17/18	3.2613	3.3362	231	14/18	3.1087	3.1805	472	10/18	2.9095	2.9828	472	13/18	2.7461	2.8218
231	8/18	3.2696	3.3381	231	15/18	3.1115	3.1833	472	11/18	2.9035	2.9773	472	14/18	2.7479	2.8238
231	9/18	3.2542	3.3229	231	16/18	3.1245	3.1969	472	12/18	2.8931	2.9684	472	15/18	2.7556	2.8310
231	10/18	3.2462	3.3155	336	8/20	3.2038	3.2751	472	13/18	2.8959	2.9699	472	16/18	2.7655	2.8410
231	11/18	3.2469	3.3167	336	9/20	3.1698	3.2396	472	14/18	2.8923	2.9662	472	17/18	2.7836	2.8612
231	12/18	3.2532	3.3213	336	10/20	3.1660	3.2364	472	15/18	2.8961	2.9701	666	10/21	2.7918	2.8656
231	13/18	3.2479	3.3174	336	11/20	3.1764	3.2456	472	16/18	2.9023	2.9772	666	11/21	2.7846	2.8589
				336	12/20	3.1707	3.2424	472	17/18	2.9165	2.9923	666	12/21	2.7810	2.8556

Table 21: Decoder-depth sweep of E2D2 $b=16$ (part 2 of 2): every trained model, one row per decoder depth.

N	L_{dec}/L	DUEL	ELBO
$C = 6 \times 10^{19}$ (cont.)			
666	13/21	2.7724	2.8466
666	14/21	2.7700	2.8447
666	15/21	2.7675	2.8432
666	16/21	2.7751	2.8503
666	17/21	2.7710	2.8465
666	18/21	2.7764	2.8519
666	19/21	2.7817	2.8579
666	20/21	2.8005	2.8772
944	14/25	2.8181	2.8915
944	15/25	2.8161	2.8914
944	16/25	2.8265	2.9005
944	17/25	2.8104	2.8834
944	18/25	2.8149	2.8889
944	19/25	2.8134	2.8879
944	20/25	2.8107	2.8862
944	21/25	2.8131	2.8866
944	22/25	2.8142	2.8881
944	23/25	2.8184	2.8938
944	24/25	2.8282	2.9038
$C = 10^{20}$			
206	7/16	2.7813	2.8535
206	8/16	2.7601	2.8330
206	9/16	2.7437	2.8164
206	10/16	2.7314	2.8049
206	11/16	2.7242	2.7980
206	12/16	2.7196	2.7942
206	13/16	2.7239	2.7984
206	14/16	2.7369	2.8125
206	15/16	2.7635	2.8412
268	7/16	2.7530	2.8257
268	8/16	2.7323	2.8054
268	9/16	2.7170	2.7899
268	10/16	2.7059	2.7791
268	11/16	2.6967	2.7707
268	12/16	2.6967	2.7710
268	13/16	2.6963	2.7715
268	14/16	2.7103	2.7861
268	15/16	2.7374	2.8139
336	11/20	2.7018	2.7748
336	12/20	2.6964	2.7695
336	13/20	2.6897	2.7632
336	14/20	2.6812	2.7551
336	15/20	2.6786	2.7521
336	16/20	2.6788	2.7528
336	17/20	2.6840	2.7582
336	18/20	2.7013	2.7771
336	19/20	2.7217	2.8012
472	8/18	2.7056	2.7798
472	9/18	2.6917	2.7663
472	10/18	2.6821	2.7563
472	11/18	2.6755	2.7492
472	12/18	2.6705	2.7455
472	13/18	2.6678	2.7422
472	14/18	2.6689	2.7430
472	15/18	2.6709	2.7465
472	16/18	2.6895	2.7653
472	17/18	2.7068	2.7839
666	11/21	2.6899	2.7644
666	12/21	2.6825	2.7573
666	13/21	2.6794	2.7534
666	14/21	2.6712	2.7462
666	15/21	2.6698	2.7448
666	16/21	2.6702	2.7444
666	17/21	2.6700	2.7444
666	18/21	2.6800	2.7551

N	L_{dec}/L	DUEL	ELBO
$C = 10^{20}$ (cont.)			
666	19/21	2.6891	2.7651
666	20/21	2.7072	2.7864
944	14/25	2.7049	2.7792
944	15/25	2.6998	2.7757
944	16/25	2.6951	2.7698
944	17/25	2.6902	2.7635
944	18/25	2.6903	2.7651
944	19/25	2.6918	2.7668
944	20/25	2.6901	2.7645
944	21/25	2.6979	2.7731
944	22/25	2.6994	2.7749
944	23/25	2.7136	2.7899
944	24/25	2.7256	2.8027
1233	14/24	2.7341	2.8092
1233	15/24	2.7299	2.8049
1233	16/24	2.7303	2.8054
1233	17/24	2.7345	2.8089
1233	19/24	2.7302	2.8051
1233	20/24	2.7258	2.8012
1233	21/24	2.7311	2.8071
1233	22/24	2.7358	2.8117
1233	23/24	2.7430	2.8203
$C = 3 \times 10^{20}$			
831	15/22	2.5136	2.5862
831	16/22	2.5095	2.5824
831	17/22	2.5097	2.5834
944	16/25	2.5127	2.5858
944	17/25	2.5123	2.5852
944	18/25	2.5106	2.5841
944	19/25	2.5084	2.5816
944	20/25	2.5112	2.5838
1233	16/24	2.5199	2.5930
1233	17/24	2.5165	2.5897
1233	18/24	2.5129	2.5866

Table 22: Decoder-depth sweep of E2D2 $b=8$ (part 1 of 2): every trained model, one row per decoder depth. N is the nominal model size in millions of non-embedding parameters, as named in Table 6; DUEL and ELBO are validation NLLs in nats per token. L_{dec}/L is the decoder depth over the total depth. The best L_{dec} of each (budget, size) group is in bold.

N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO
$C = 6 \times 10^{18}$				$C = 10^{19}$ (cont.)				$C = 3 \times 10^{19}$ (cont.)				$C = 6 \times 10^{19}$ (cont.)			
34	2/8	3.3391	3.4108	85	6/13	3.0641	3.1339	206	14/16	2.8432	2.9188	472	11/18	2.6989	2.7701
34	3/8	3.2682	3.3356	85	7/13	3.0438	3.1147	206	15/16	2.8737	2.9502	472	12/18	2.6987	2.7704
34	4/8	3.2355	3.3051	85	8/13	3.0353	3.1070	268	8/16	2.8263	2.8980	472	13/18	2.6969	2.7691
34	5/8	3.2167	3.2879	85	9/13	3.0327	3.1051	268	9/16	2.8153	2.8880	472	14/18	2.6968	2.7694
34	6/8	3.2092	3.2822	85	10/13	3.0409	3.1141	268	10/16	2.8107	2.8831	472	15/18	2.7079	2.7810
34	7/8	3.2386	3.3156	85	11/13	3.0553	3.1286	268	11/16	2.8074	2.8800	472	16/18	2.7273	2.8019
48	2/9	3.3077	3.3766	85	12/13	3.0869	3.1637	268	12/16	2.8076	2.8812	666	10/21	2.7268	2.7985
48	3/9	3.2392	3.3078	113	4/12	3.0764	3.1453	268	13/16	2.8168	2.8911	666	11/21	2.7237	2.7955
48	4/9	3.1997	3.2707	113	5/12	3.0560	3.1259	268	14/16	2.8390	2.9148	666	12/21	2.7173	2.7893
48	5/9	3.1757	3.2461	113	6/12	3.0334	3.1041	268	15/16	2.8663	2.9437	666	13/21	2.7128	2.7848
48	6/9	3.1698	3.2405	113	7/12	3.0233	3.0947	336	10/20	2.8274	2.8999	666	14/21	2.7133	2.7860
48	7/9	3.1739	3.2472	113	8/12	3.0175	3.0900	336	11/20	2.8230	2.8957	666	15/21	2.7093	2.7812
48	8/9	3.2048	3.2808	113	9/12	3.0266	3.1011	336	12/20	2.8162	2.8900	666	16/21	2.7159	2.7885
66	2/10	3.2886	3.3568	113	10/12	3.0477	3.1232	336	14/20	2.8119	2.8847	666	17/21	2.7181	2.7915
66	3/10	3.2121	3.2812	113	11/12	3.0709	3.1476	336	15/20	2.8088	2.8823	666	18/21	2.7265	2.8007
66	4/10	3.1750	3.2441	142	6/15	3.0496	3.1206	336	16/20	2.8138	2.8881	944	14/25	2.7562	2.8282
66	5/10	3.1579	3.2267	142	7/15	3.0376	3.1081	336	17/20	2.8354	2.9105	944	15/25	2.7547	2.8281
66	6/10	3.1460	3.2171	142	8/15	3.0256	3.0972	336	18/20	2.8474	2.9229	944	16/25	2.7626	2.8352
66	7/10	3.1412	3.2141	142	9/15	3.0230	3.0955	472	8/18	2.8496	2.9215	944	17/25	2.7568	2.8289
66	8/10	3.1604	3.2346	142	10/15	3.0159	3.0883	472	9/18	2.8450	2.9175	944	18/25	2.7596	2.8330
66	9/10	3.1849	3.2623	142	11/15	3.0156	3.0892	472	10/18	2.8488	2.9204	944	19/25	2.7524	2.8256
85	4/13	3.1946	3.2630	142	12/15	3.0244	3.0975	472	11/18	2.8430	2.9152	944	20/25	2.7576	2.8302
85	5/13	3.1726	3.2414	142	13/15	3.0530	3.1292	472	12/18	2.8424	2.9155	944	21/25	2.7526	2.8254
85	6/13	3.1515	3.2232	142	14/15	3.0762	3.1545	472	13/18	2.8368	2.9100	944	22/25	2.7660	2.8395
85	7/13	3.1352	3.2068	170	9/18	3.0391	3.1105	472	14/18	2.8398	2.9137	$C = 10^{20}$			
85	8/13	3.1308	3.2015	170	10/18	3.0349	3.1074	472	15/18	2.8455	2.9193	268	8/16	2.6807	2.7504
85	9/13	3.1310	3.2033	170	11/18	3.0246	3.0965	472	16/18	2.8568	2.9323	268	9/16	2.6647	2.7343
85	10/13	3.1368	3.2112	170	12/18	3.0239	3.0970	666	11/21	2.8856	2.9580	268	10/16	2.6562	2.7263
85	11/13	3.1533	3.2274	170	13/18	3.0192	3.0934	666	12/21	2.8856	2.9578	268	11/16	2.6501	2.7208
85	12/13	3.1849	3.2621	170	14/18	3.0231	3.0968	666	13/21	2.8854	2.9581	268	12/16	2.6507	2.7220
113	4/12	3.1760	3.2457	170	15/18	3.0341	3.1097	666	14/21	2.8898	2.9636	268	13/16	2.6535	2.7258
113	5/12	3.1503	3.2205	170	16/18	3.0595	3.1353	666	15/21	2.8914	2.9639	268	14/16	2.6693	2.7427
113	6/12	3.1394	3.2100	170	17/18	3.0878	3.1659	666	16/21	2.8771	2.9501	268	15/16	2.6986	2.7742
113	7/12	3.1355	3.2064	231	8/18	3.0534	3.1251	666	17/21	2.8802	2.9533	336	11/20	2.6546	2.7243
113	8/12	3.1259	3.1987	231	9/18	3.0469	3.1189	666	18/21	2.8856	2.9595	336	12/20	2.6476	2.7173
113	9/12	3.1424	3.2152	231	10/18	3.0599	3.1326	666	19/21	2.8983	2.9728	336	13/20	2.6383	2.7088
113	10/12	3.1649	3.2403	231	11/18	3.0374	3.1105	$C = 6 \times 10^{19}$				336	14/20	2.6344	2.7046
113	11/12	3.1820	3.2602	231	12/18	3.0564	3.1292	206	8/16	2.7552	2.8238	336	15/20	2.6314	2.7019
170	8/18	3.1673	3.2381	231	13/18	3.0323	3.1063	206	9/16	2.7430	2.8130	336	16/20	2.6364	2.7077
170	9/18	3.1587	3.2310	231	14/18	3.0554	3.1295	206	10/16	2.7341	2.8043	336	17/20	2.6425	2.7146
170	10/18	3.1561	3.2286	231	15/18	3.0544	3.1291	206	11/16	2.7287	2.7993	336	18/20	2.6615	2.7347
170	11/18	3.1505	3.2231	231	16/18	3.0803	3.1553	206	12/16	2.7271	2.7978	336	19/20	2.6919	2.7687
170	12/18	3.1467	3.2190	336	8/20	3.1034	3.1759	206	13/16	2.7305	2.8030	472	8/18	2.6443	2.7125
170	13/18	3.1457	3.2187	336	9/20	3.1241	3.1958	206	14/16	2.7488	2.8223	472	9/18	2.6355	2.7043
170	14/18	3.1504	3.2236	336	10/20	3.1047	3.1780	206	15/16	2.7773	2.8535	472	10/18	2.6285	2.6968
170	15/18	3.1657	3.2402	336	11/20	3.1223	3.1948	268	8/16	2.7310	2.8010	472	11/18	2.6225	2.6924
170	16/18	3.1883	3.2649	336	12/20	3.1077	3.1805	268	9/16	2.7200	2.7905	472	12/18	2.6168	2.6868
231	7/18	3.2318	3.3039	336	13/20	3.1074	3.1810	268	10/16	2.7131	2.7838	472	13/18	2.6173	2.6874
231	8/18	3.1947	3.2657	336	14/20	3.1072	3.1809	268	11/16	2.7076	2.7785	472	14/18	2.6215	2.6924
231	9/18	3.1851	3.2572	336	15/20	3.1021	3.1754	268	12/16	2.7096	2.7817	472	15/18	2.6283	2.6999
231	10/18	3.2050	3.2760	336	16/20	3.0884	3.1624	268	13/16	2.7145	2.7867	666	10/21	2.6385	2.7078
231	11/18	3.1760	3.2481	$C = 3 \times 10^{19}$				268	14/16	2.7316	2.8056	666	11/21	2.6330	2.7033
231	12/18	3.2084	3.2811	142	8/15	2.8637	2.9332	268	15/16	2.7632	2.8391	666	12/21	2.6277	2.6985
231	13/18	3.1670	3.2395	142	9/15	2.8489	2.9200	336	10/20	2.7247	2.7952	666	13/21	2.6231	2.6938
231	14/18	3.1914	3.2634	142	10/15	2.8438	2.9155	336	11/20	2.7152	2.7864	666	14/21	2.6219	2.6927
231	15/18	3.1962	3.2710	142	11/15	2.8403	2.9125	336	12/20	2.7047	2.7760	666	15/21	2.6207	2.6913
$C = 10^{19}$				142	12/15	2.8448	2.9179	336	13/20	2.7038	2.7756	666	16/21	2.6181	2.6896
66	3/10	3.1454	3.2138	142	13/15	2.8587	2.9340	336	14/20	2.7007	2.7721	666	17/21	2.6222	2.6933
66	4/10	3.0988	3.1682	142	14/15	2.8905	2.9664	336	15/20	2.6986	2.7710	666	18/21	2.6310	2.7030
66	5/10	3.0779	3.1472	206	8/16	2.8349	2.9046	336	16/20	2.7011	2.7736	944	14/25	2.6424	2.7135
66	6/10	3.0635	3.1340	206	9/16	2.8247	2.8958	336	17/20	2.7102	2.7844	944	15/25	2.6393	2.7104
66	7/10	3.0599	3.1317	206	10/16	2.8155	2.8867	336	18/20	2.7297	2.8034	944	16/25	2.6385	2.7104
66	8/10	3.0652	3.1388	206	11/16	2.8150	2.8880	472	8/18	2.7219	2.7924	944	17/25	2.6362	2.7074
66	9/10	3.0973	3.1735	206	12/16	2.8161	2.8882	472	9/18	2.7141	2.7850	944	18/25	2.6357	2.7074
85	5/13	3.0832	3.1518	206	13/16	2.8233	2.8969	472	10/18	2.7066	2.7771	944	19/25	2.6357	2.7082

Table 23: Decoder-depth sweep of E2D2 $b=8$ (part 2 of 2): every trained model, one row per decoder depth.

N	L_{dec}/L	DUEL	ELBO
$C = 10^{20}$ (cont.)			
944	20/25	2.6426	2.7151
944	21/25	2.6398	2.7117
944	22/25	2.6566	2.7302
1233	12/24	2.6782	2.7507
1233	13/24	2.6743	2.7464
1233	14/24	2.6712	2.7427
1233	15/24	2.6751	2.7469
1233	16/24	2.6657	2.7384
1233	17/24	2.6696	2.7422
1233	18/24	2.6692	2.7415
1233	19/24	2.6730	2.7449
1233	20/24	2.6758	2.7491

Table 24: Decoder-depth sweep of E2D2 $b=4$ (part 1 of 3): every trained model, one row per decoder depth. N is the nominal model size in millions of non-embedding parameters, as named in Table 6; DUEL and ELBO are validation NLLs in nats per token. L_{dec}/L is the decoder depth over the total depth. The best L_{dec} of each (budget, size) group is in bold.

N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO
$C = 6 \times 10^{18}$				$C = 6 \times 10^{18}$ (cont.)				$C = 10^{19}$ (cont.)				$C = 10^{19}$ (cont.)			
34	2/8	3.2262	3.2826	231	8/18	3.0854	3.1464	113	9/12	2.9466	3.0098	472	9/18	3.0480	3.1097
34	3/8	3.1761	3.2335	231	9/18	3.0870	3.1500	113	10/12	2.9768	3.0424	472	10/18	3.0556	3.1181
34	4/8	3.1530	3.2133	231	10/18	3.0777	3.1400	113	11/12	3.0116	3.0795	472	11/18	3.0455	3.1082
34	5/8	3.1417	3.2042	231	11/18	3.0780	3.1398	142	3/15	3.0189	3.0731	472	12/18	3.0566	3.1194
34	6/8	3.1368	3.2019	231	12/18	3.0774	3.1416	142	4/15	2.9882	3.0437	472	13/18	3.0434	3.1064
34	7/8	3.1803	3.2489	231	13/18	3.0898	3.1523	142	5/15	2.9670	3.0239	472	14/18	3.0454	3.1091
48	2/9	3.1894	3.2444	231	14/18	3.0908	3.1542	142	6/15	2.9550	3.0121	472	15/18	3.0544	3.1185
48	3/9	3.1356	3.1932	231	15/18	3.1165	3.1812	142	7/15	2.9458	3.0045	472	16/18	3.0602	3.1252
48	4/9	3.1095	3.1675	231	16/18	3.1464	3.2133	142	8/15	2.9367	2.9971	472	17/18	3.0848	3.1521
48	5/9	3.0923	3.1527	231	17/18	3.2015	3.2731	142	9/15	2.9310	2.9918	$C = 3 \times 10^{19}$			
48	6/9	3.0941	3.1554	336	2/20	3.2797	3.3339	142	10/15	2.9333	2.9950	142	4/15	2.8474	2.9014
48	7/9	3.0999	3.1653	336	3/20	3.2703	3.3282	142	11/15	2.9362	2.9989	142	5/15	2.8215	2.8772
48	8/9	3.1420	3.2100	336	4/20	3.2812	3.3417	142	12/15	2.9478	3.0116	142	6/15	2.8036	2.8593
66	2/10	3.1607	3.2162	336	5/20	3.2417	3.3034	142	13/15	2.9769	3.0414	142	7/15	2.7955	2.8525
66	3/10	3.1083	3.1649	336	6/20	3.2278	3.2902	142	14/15	3.0164	3.0850	142	8/15	2.7849	2.8422
66	4/10	3.0793	3.1374	336	7/20	3.2010	3.2630	170	4/18	3.0089	3.0648	142	9/15	2.7756	2.8333
66	5/10	3.0672	3.1266	336	8/20	3.1415	3.2038	170	5/18	2.9824	3.0404	142	10/15	2.7756	2.8343
66	6/10	3.0664	3.1270	336	9/20	3.2137	3.2766	170	6/18	2.9722	3.0308	142	11/15	2.7737	2.8335
66	7/10	3.0702	3.1322	336	10/20	3.1313	3.1947	170	7/18	2.9595	3.0183	142	12/15	2.7809	2.8425
66	8/10	3.0798	3.1442	336	11/20	3.1955	3.2597	170	8/18	2.9469	3.0065	142	13/15	2.7978	2.8602
66	9/10	3.1249	3.1930	336	12/20	3.1408	3.2052	170	9/18	2.9431	3.0036	142	14/15	2.8355	2.9001
85	2/13	3.1711	3.2241	336	13/20	3.1485	3.2123	170	10/18	2.9403	3.0018	206	4/16	2.8121	2.8658
85	3/13	3.1202	3.1756	336	14/20	3.1275	3.1904	170	11/18	2.9358	2.9971	206	5/16	2.7933	2.8480
85	4/13	3.0906	3.1462	336	15/20	3.1393	3.2036	170	12/18	2.9349	2.9963	206	6/16	2.7719	2.8277
85	5/13	3.0702	3.1279	336	16/20	3.1700	3.2349	170	13/18	2.9362	2.9978	206	8/16	2.7535	2.8107
85	6/13	3.0552	3.1155	336	17/20	3.1777	3.2436	170	14/18	2.9442	3.0074	206	9/16	2.7487	2.8065
85	7/13	3.0463	3.1065	336	18/20	3.1915	3.2589	170	15/18	2.9531	3.0177	206	10/16	2.7475	2.8059
85	8/13	3.0448	3.1057	336	19/20	3.2588	3.3340	170	16/18	2.9886	3.0553	206	11/16	2.7463	2.8057
85	9/13	3.0495	3.1125	472	2/18	3.2975	3.3505	170	17/18	3.0301	3.0994	206	12/16	2.7454	2.8046
85	10/13	3.0595	3.1235	472	3/18	3.2595	3.3149	231	4/18	3.0219	3.0788	206	13/16	2.7563	2.8177
85	11/13	3.0826	3.1488	472	4/18	3.2478	3.3063	231	5/18	2.9889	3.0464	206	14/16	2.7814	2.8446
85	12/13	3.1251	3.1932	472	5/18	3.2865	3.3474	231	6/18	2.9789	3.0384	206	15/16	2.8161	2.8806
113	2/12	3.1474	3.2003	472	6/18	3.2411	3.3019	231	7/18	2.9686	3.0279	268	3/16	2.8260	2.8783
113	3/12	3.0970	3.1530	472	7/18	3.2493	3.3107	231	8/18	2.9557	3.0164	268	4/16	2.7929	2.8464
113	4/12	3.0689	3.1266	472	8/18	3.2467	3.3073	231	9/18	2.9529	3.0139	268	5/16	2.7751	2.8296
113	5/12	3.0507	3.1095	472	9/18	3.2641	3.3274	231	10/18	2.9477	3.0086	268	6/16	2.7559	2.8117
113	6/12	3.0410	3.1007	472	10/18	3.2269	3.2898	231	11/18	2.9426	3.0043	268	7/16	2.7492	2.8060
113	7/12	3.0360	3.0984	472	11/18	3.2243	3.2877	231	12/18	2.9434	3.0063	268	8/16	2.7430	2.8008
113	8/12	3.0362	3.0979	472	12/18	3.2456	3.3094	231	13/18	2.9521	3.0148	268	9/16	2.7340	2.7923
113	9/12	3.0549	3.1200	472	13/18	3.2295	3.2934	231	14/18	2.9487	3.0114	268	10/16	2.7338	2.7927
113	10/12	3.0934	3.1587	472	14/18	3.2243	3.2883	231	15/18	2.9779	3.0417	268	11/16	2.7317	2.7905
113	11/12	3.1255	3.1943	472	15/18	3.2244	3.2891	231	16/18	3.0055	3.0704	268	12/16	2.7392	2.7993
170	2/18	3.2066	3.2599	472	16/18	3.2368	3.3033	231	17/18	3.0416	3.1102	268	13/16	2.7479	2.8095
170	3/18	3.1560	3.2116	472	17/18	3.2727	3.3419	336	4/20	3.0964	3.1552	268	14/16	2.7724	2.8356
170	4/18	3.1256	3.1829	$C = 10^{19}$				336	5/20	3.0726	3.1328	268	15/16	2.8097	2.8743
170	5/18	3.1012	3.1596	66	4/10	3.0106	3.0676	336	6/20	3.0382	3.0987	336	4/20	2.8114	2.8660
170	6/18	3.0895	3.1490	66	5/10	2.9942	3.0531	336	7/20	3.0004	3.0602	336	5/20	2.7910	2.8457
170	7/18	3.0757	3.1357	66	6/10	2.9908	3.0510	336	8/20	2.9888	3.0497	336	6/20	2.7713	2.8258
170	8/18	3.0676	3.1289	66	7/10	2.9894	3.0517	336	9/20	2.9890	3.0501	336	7/20	2.7625	2.8192
170	9/18	3.0624	3.1246	66	8/10	2.9954	3.0599	336	10/20	2.9805	3.0415	336	8/20	2.7518	2.8086
170	10/18	3.0621	3.1243	66	9/10	3.0359	3.1031	336	11/20	2.9809	3.0426	336	9/20	2.7465	2.8044
170	11/18	3.0536	3.1162	85	4/13	3.0153	3.0710	336	12/20	2.9914	3.0548	336	10/20	2.7380	2.7957
170	12/18	3.0548	3.1177	85	5/13	2.9904	3.0473	336	13/20	2.9982	3.0606	336	11/20	2.7376	2.7961
170	13/18	3.0553	3.1181	85	6/13	2.9753	3.0333	336	14/20	2.9899	3.0530	336	12/20	2.7352	2.7946
170	14/18	3.0590	3.1224	85	7/13	2.9639	3.0233	336	15/20	2.9965	3.0603	336	13/20	2.7353	2.7950
170	15/18	3.0733	3.1380	85	8/13	2.9567	3.0169	336	16/20	3.0084	3.0718	336	14/20	2.7338	2.7938
170	16/18	3.1153	3.1829	85	9/13	2.9661	3.0271	336	17/20	3.0198	3.0847	336	15/20	2.7378	2.7979
170	17/18	3.1604	3.2310	85	10/13	2.9705	3.0337	336	18/20	3.0396	3.1058	336	16/20	2.7419	2.8029
231	2/18	3.2208	3.2750	85	11/13	2.9839	3.0494	336	19/20	3.0807	3.1524	336	17/20	2.7562	2.8188
231	3/18	3.1737	3.2307	85	12/13	3.0265	3.0932	472	4/18	3.0902	3.1475	336	18/20	2.7837	2.8482
231	4/18	3.1396	3.1971	113	4/12	2.9733	3.0294	472	5/18	3.0874	3.1465	336	19/20	2.8215	2.8893
231	5/18	3.1229	3.1814	113	5/12	2.9566	3.0139	472	6/18	3.0325	3.0915	472	4/18	2.8030	2.8573
231	6/18	3.1165	3.1768	113	6/12	2.9422	3.0005	472	7/18	3.0649	3.1255	472	5/18	2.7847	2.8403
231	7/18	3.0962	3.1571	113	7/12	2.9395	2.9993	472	8/18	3.0558	3.1163	472	6/18	2.7723	2.8282
				113	8/12	2.9363	2.9970	472				472	7/18	2.7641	2.8215

Table 25: Decoder-depth sweep of E2D2 $b=4$ (part 2 of 3): every trained model, one row per decoder depth.

N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO	N	L_{dec}/L	DUEL	ELBO
$C = 3 \times 10^{19}$ (cont.)				$C = 6 \times 10^{19}$ (cont.)				$C = 6 \times 10^{19}$ (cont.)				$C = 10^{20}$ (cont.)			
472	8/18	2.7557	2.8136	268	9/16	2.6497	2.7047	944	14/25	2.6712	2.7289	472	11/18	2.5524	2.6063
472	9/18	2.7522	2.8107	268	10/16	2.6419	2.6981	944	15/25	2.6678	2.7253	472	12/18	2.5555	2.6107
472	10/18	2.7473	2.8059	268	11/16	2.6379	2.6944	944	16/25	2.6762	2.7350	472	13/18	2.5541	2.6096
472	11/18	2.7401	2.7998	268	12/16	2.6453	2.7028	944	17/25	2.6729	2.7318	472	14/18	2.5601	2.6164
472	12/18	2.7457	2.8048	268	13/16	2.6531	2.7117	944	18/25	2.6690	2.7277	472	15/18	2.5693	2.6266
472	13/18	2.7531	2.8128	268	14/16	2.6728	2.7333	944	19/25	2.6664	2.7255	472	16/18	2.5909	2.6503
472	14/18	2.7558	2.8160	336	4/20	2.7227	2.7742	944	20/25	2.6770	2.7357	472	17/18	2.6229	2.6840
472	15/18	2.7699	2.8313	336	5/20	2.7075	2.7609	944	21/25	2.6759	2.7341	472	17/18	2.6229	2.6840
472	16/18	2.7879	2.8507	336	6/20	2.6809	2.7342	944	22/25	2.6891	2.7493	666	3/21	2.6577	2.7067
472	17/18	2.8122	2.8768	336	7/20	2.6661	2.7203	944	23/25	2.7030	2.7643	666	4/21	2.6229	2.6728
666	4/21	2.8589	2.9140	336	8/20	2.6553	2.7095	944	24/25	2.7255	2.7886	666	5/21	2.6037	2.6549
666	5/21	2.8463	2.9021	336	9/20	2.6556	2.7107	1233	4/24	2.7630	2.8180	666	6/21	2.5892	2.6415
666	6/21	2.8193	2.8759	336	10/20	2.6363	2.6920	1233	5/24	2.7853	2.8420	666	7/21	2.5738	2.6265
666	7/21	2.8202	2.8785	336	11/20	2.6368	2.6930	1233	6/24	2.7545	2.8118	666	8/21	2.5668	2.6199
666	8/21	2.8150	2.8733	336	12/20	2.6355	2.6918	1233	7/24	2.7437	2.8015	666	9/21	2.5576	2.6114
666	9/21	2.7979	2.8566	336	13/20	2.6365	2.6933	1233	8/24	2.7343	2.7917	666	10/21	2.5535	2.6074
666	10/21	2.7952	2.8535	336	14/20	2.6336	2.6904	1233	9/24	2.7298	2.7875	666	11/21	2.5515	2.6056
666	11/21	2.7941	2.8533	336	15/20	2.6367	2.6945	1233	10/24	2.7349	2.7935	666	12/21	2.5506	2.6055
666	12/21	2.7961	2.8562	336	16/20	2.6379	2.6964	1233	11/24	2.7293	2.7880	666	13/21	2.5488	2.6039
666	13/21	2.7839	2.8438	336	17/20	2.6494	2.7091	1233	12/24	2.7199	2.7780	666	14/21	2.5484	2.6042
666	14/21	2.7901	2.8500	336	18/20	2.6719	2.7326	1233	13/24	2.7287	2.7877	666	15/21	2.5492	2.6049
666	15/21	2.7911	2.8510	336	19/20	2.7075	2.7724	1233	14/24	2.7186	2.7777	666	16/21	2.5535	2.6098
666	16/21	2.7971	2.8580	472	3/18	2.7143	2.7648	1233	15/24	2.7141	2.7731	666	17/21	2.5562	2.6130
666	17/21	2.7984	2.8591	472	4/18	2.6872	2.7388	1233	16/24	2.7417	2.8025	666	18/21	2.5697	2.6275
666	18/21	2.8061	2.8683	472	5/18	2.6659	2.7180	1233	17/24	2.7216	2.7817	666	19/21	2.5902	2.6495
666	19/21	2.8174	2.8795	472	6/18	2.6506	2.7037	1233	18/24	2.7215	2.7806	666	20/21	2.6251	2.6870
666	20/21	2.8400	2.9027	472	7/18	2.6413	2.6953	1233	19/24	2.7165	2.7751	944	4/25	2.6421	2.6934
944	4/25	2.9194	2.9758	472	8/18	2.6340	2.6891	1233	20/24	2.7208	2.7804	944	5/25	2.6204	2.6729
944	5/25	2.9191	2.9768	472	9/18	2.6270	2.6821	1233	21/24	2.7187	2.7782	944	6/25	2.6076	2.6604
944	6/25	2.9078	2.9666	472	10/18	2.6227	2.6780	1233	22/24	2.7286	2.7885	944	7/25	2.5954	2.6487
944	7/25	2.8925	2.9508	472	11/18	2.6206	2.6773	1233	23/24	2.7476	2.8087	944	8/25	2.5867	2.6406
944	8/25	2.8742	2.9344	472	12/18	2.6223	2.6796	$C = 10^{20}$				944	9/25	2.5767	2.6317
944	9/25	2.8795	2.9398	472	13/18	2.6214	2.6791	268	5/16	2.6477	2.7001	944	10/25	2.5714	2.6258
944	10/25	2.8697	2.9289	472	14/18	2.6276	2.6855	268	6/16	2.6346	2.6875	944	11/25	2.5679	2.6234
944	11/25	2.9058	2.9659	472	15/18	2.6405	2.6996	268	7/16	2.6178	2.6726	944	12/25	2.5690	2.6247
944	12/25	2.8689	2.9293	472	16/18	2.6639	2.7245	268	8/16	2.6048	2.6591	944	13/25	2.5633	2.6185
944	13/25	2.8468	2.9071	472	17/18	2.6940	2.7561	268	9/16	2.5947	2.6495	944	14/25	2.5622	2.6181
944	14/25	2.8795	2.9402	472	18/18			268	10/16	2.5910	2.6467	944	15/25	2.5610	2.6171
944	15/25	2.8439	2.9051	666	4/21	2.7031	2.7556	268	11/16	2.5853	2.6405	944	16/25	2.5630	2.6200
944	16/25	2.8612	2.9219	666	5/21	2.6807	2.7336	268	12/16	2.5890	2.6454	944	17/25	2.5609	2.6169
944	17/25	2.8485	2.9108	666	6/21	2.6704	2.7245	268	13/16	2.5966	2.6543	944	18/25	2.5644	2.6204
944	18/25	2.8409	2.9036	666	7/21	2.6612	2.7163	268	14/16	2.6139	2.6734	944	19/25	2.5634	2.6199
944	19/25	2.8564	2.9181	666	8/21	2.6494	2.7043	336	4/20	2.6765	2.7279	944	20/25	2.5651	2.6215
944	20/25	2.8542	2.9160	666	9/21	2.6435	2.6997	336	5/20	2.6583	2.7109	944	21/25	2.5707	2.6278
944	21/25	2.8547	2.9168	666	10/21	2.6377	2.6934	336	6/20	2.6362	2.6898	944	22/25	2.5842	2.6421
944	22/25	2.8533	2.9162	666	11/21	2.6380	2.6945	336	7/20	2.6168	2.6706	944	23/25	2.6057	2.6642
944	23/25	2.8615	2.9240	666	12/21	2.6407	2.6984	336	8/20	2.6108	2.6651	944	24/25	2.6330	2.6955
944	24/25	2.8812	2.9459	666	13/21	2.6332	2.6901	336	9/20	2.6024	2.6564	1233	4/24	2.6590	2.7112
$C = 6 \times 10^{19}$				666	14/21	2.6346	2.6925	336	10/20	2.5878	2.6427	1233	5/24	2.6434	2.6964
206	4/16	2.7523	2.8055	666	15/21	2.6313	2.6887	336	11/20	2.5856	2.6406	1233	6/24	2.6366	2.6912
206	5/16	2.7298	2.7842	666	16/21	2.6321	2.6904	336	12/20	2.5774	2.6323	1233	7/24	2.6208	2.6751
206	6/16	2.7063	2.7607	666	17/21	2.6384	2.6965	336	13/20	2.5744	2.6300	1233	8/24	2.6089	2.6640
206	7/16	2.6884	2.7432	666	18/21	2.6578	2.7169	336	14/20	2.5728	2.6287	1233	9/24	2.6036	2.6589
206	8/16	2.6833	2.7385	666	19/21	2.6749	2.7363	336	15/20	2.5724	2.6292	1233	10/24	2.6059	2.6619
206	9/16	2.6707	2.7269	666	20/21	2.7062	2.7701	336	16/20	2.5765	2.6338	1233	11/24	2.5904	2.6469
206	10/16	2.6645	2.7214	944	4/25	2.7380	2.7908	336	17/20	2.5849	2.6432	1233	12/24	2.5864	2.6419
206	11/16	2.6620	2.7193	944	5/25	2.7249	2.7800	336	18/20	2.6046	2.6640	1233	13/24	2.5858	2.6416
206	12/16	2.6618	2.7200	944	6/25	2.7171	2.7731	336	19/20	2.6394	2.7017	1233	14/24	2.5971	2.6543
206	13/16	2.6696	2.7285	944	7/25	2.6976	2.7531	472	4/18	2.6306	2.6810	1233	15/24	2.5936	2.6502
206	14/16	2.6917	2.7526	944	8/25	2.6959	2.7531	472	5/18	2.5997	2.6506	1233	16/24	2.5918	2.6488
268	4/16	2.7199	2.7725	944	9/25	2.6984	2.7552	472	6/18	2.5820	2.6330	1233	17/24	2.5972	2.6541
268	5/16	2.6993	2.7534	944	10/25	2.6978	2.7553	472	7/18	2.5704	2.6227	1233	18/24	2.5922	2.6492
268	6/16	2.6735	2.7265	944	11/25	2.6845	2.7424	472	8/18	2.5614	2.6141	1233	19/24	2.5931	2.6502
268	7/16	2.6604	2.7144	944	12/25	2.6793	2.7376	472	9/18	2.5581	2.6111	1233	20/24	2.5928	2.6498
268	8/16	2.6531	2.7081	944	13/25	2.6772	2.7355	472	10/18	2.5538	2.6079	1233	21/24	2.6052	2.6632

Table 26: Decoder-depth sweep of E2D2 $b=4$ (part 3 of 3): every trained model, one row per decoder depth.

N	L_{dec}/L	DUEL	ELBO
$C = 10^{20}$ (cont.)			
1233	22/24	2.6232	2.6821
1233	23/24	2.6418	2.7035
1678	4/25	2.6946	2.7491
1678	5/25	2.6936	2.7485
1678	6/25	2.6715	2.7271
1678	7/25	2.6638	2.7204
1678	8/25	2.6658	2.7217
1678	9/25	2.6566	2.7136
1678	10/25	2.6458	2.7022
1678	11/25	2.6635	2.7211
1678	12/25	2.6448	2.7016
1678	13/25	2.6374	2.6945
1678	14/25	2.6472	2.7049
1678	15/25	2.6409	2.6976
1678	16/25	2.6394	2.6973
1678	17/25	2.6477	2.7055
1678	18/25	2.6475	2.7057
1678	19/25	2.6486	2.7068
1678	20/25	2.6484	2.7068
1678	21/25	2.6397	2.6976
1678	22/25	2.6442	2.7034
1678	23/25	2.6571	2.7165
1678	24/25	2.6741	2.7344