BEYOND AUTOREGRESSIVE RECIPES: SCALING UP MIXTURE-OF-EXPERTS DIFFUSION LLMS TO 30B

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion large language models (dLLMs) are a promising alternative to autoregressive (AR) models, but existing sparse Mixture-of-Experts (MoE) dLLMs largely inherit AR recipes, and the masked denoising objective calls for dLLM-specific scaling laws. We present a systematic study of scaling laws for MoE dLLMs, spanning optimization hyperparameters, compute allocation, and MoE architecture across a broad range of compute budgets. Guided by these laws, we train LLaDA MoE v2, a 30B-A3B dLLM, from scratch on 23.5T tokens; it surpasses the previous work LLaDA MoE at only half of the latter's training FLOPs. The base model approaches the strong open-weight AR MoE model Qwen3 30B-A3B on knowledge, reasoning, and coding benchmarks while using 65% as many pretraining tokens. After supervised fine-tuning, it outperforms SDAR Chat, a competitive MoE dLLM, on seven of eight reasoning and coding benchmarks. Together, these results show LLaDA MoE v2 to be the strongest dLLM trained from scratch to date, establishing dLLM-specific scaling laws as a foundation for large-scale MoE dLLM training and practical application.

1 INTRODUCTION

Diffusion language models (dLLMs) (Lou et al., 2023; Shi et al., 2024; Sahoo et al., 2024; Ou et al., 2025) have emerged as a promising alternative to autoregressive (AR) language models (Zhao et al., 2026), offering distinct advantages in test-time scaling (Ye et al., 2024; Shen et al., 2026), speculative decoding (Chen et al., 2026; Cheng et al., 2026b), unified multimodal understanding and generation (You et al., 2026a;b; Yang et al., 2026), and data utilization (Ni et al., 2025a; Prabhudesai et al., 2026). Realizing these advantages at the frontier calls for scaling dLLMs to larger sizes.

Mixture-of-Experts (MoE) architecture (Jacobs et al., 1991; Shazeer et al., 2017; Lepikhin et al., 2020; Du et al., 2022) has enabled language models to keep improving at scales beyond the reach of dense architectures, becoming standard practice in frontier models (Yang et al., 2025; Xu et al., 2026; Team et al., 2026b). MoE dLLMs have begun to emerge (Zhu et al., 2025; Feng et al., 2026; Zhang et al., 2026). Yet their scaling designs are still inherited from AR models: LLaDA MoE (Zhu et al., 2025), for instance, closely follows the AR model OLMoE (Muennighoff et al., 2025). Existing dLLM scaling studies focus on dense architectures (Nie et al., 2025; Ni et al., 2025b; von Rütte et al., 2026; Sahoo et al., 2026); how MoE dLLMs should scale remains an open question.

In this work, we present a systematic study of how MoE dLLMs scale, moving beyond inherited AR recipes to recalibrate scaling behavior and architectural design across multiple scales. We derive compute-dependent scaling laws for optimization hyperparameters and model–data allocation, and identify design principles for the three central architectural choices of MoE: routing sparsity, expert granularity, and shared-expert ratio. Together, these scaling laws and design principles translate empirical findings from small-scale sweeps into guidance for model design and large-scale training. Table 1 summarizes these findings together with the corresponding supporting evidence.

These scaling laws and principles guide the design and training of LLaDA MoE v2, a 30B-A3B MoE dLLM pretrained on 23.5T tokens. Its base model surpasses the previous LLaDA MoE (Zhu et al., 2025) with only 50% as many training FLOPs, and approaches the strong AR MoE model Qwen3 30B-A3B (Yang et al., 2025) on multiple benchmarks while using approximately 65% as many pretraining tokens. After supervised fine-tuning (SFT), it outperforms SDAR Chat 30B-A3B (Cheng et al., 2026a), a competitive MoE dLLM, on seven of eight reasoning and coding benchmarks. The

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

Table 1: **Scaling laws and design principles for MoE dLLMs.** Controlled sweeps recalibrate optimization, compute allocation, and MoE architecture for the masked denoising objective, with AR and dense-dLLM references for comparison and scale-up validation from LLaDA MoE v2.

Finding	Evidence
Optimization	
Batch size. The optimal token batch size is larger than AR laws predict.	Figs. 1, 2
Learning rate. The optimal peak value decays faster with compute than in AR.	Figs. 1, 2
Compute allocation	
Model–data split. The optimal allocation is nearly balanced, with a slight tilt toward data.	Fig. 3
Data-side tilt. MoE dLLM allocation tilts further toward data than dense dLLMs.	Table 2
MoE architecture	
Activation ratio. Lower ratios are increasingly favorable at larger compute scales.	Fig. 4a
Expert granularity. $G = 8$–16 remains near-optimal across scales.	Fig. 4b
Shared-expert ratio. $S = 33.3\%$ remains optimal across scales.	Fig. 4c
Scale-up validation	
Base model. Approaches Qwen3 30B-A3B with 65% as many pretraining tokens.	Table 3
Training efficiency. Surpasses LLaDA MoE at only half of the latter's training FLOPs.	Fig. 5
SFT. Outperforms SDAR Chat, a strong MoE dLLM baseline, on seven of eight tasks.	Table 4

results position LLaDA MoE v2 as the strongest dLLM trained from scratch to date. Together, these results close the gap left by dense-only dLLM scaling studies: they show that MoE dLLM-specific scaling laws, rather than inherited AR recipes, directly translate into effective large-scale training and a model that is competitive with strong AR systems of comparable activated size.

2 Preliminaries

2.1 Diffusion Language Models

Masked diffusion language models (Lou et al., 2023; Shi et al., 2024; Sahoo et al., 2024; Ou et al., 2025) define a discrete diffusion process over token sequences. Let $x_0 = (x_0^1, \ldots, x_0^L)$ denote a clean sequence of L tokens, and let `[MASK]` be the mask token. The forward process corrupts each position independently: at noise level $t \in [0, 1]$, the token x_0^i is replaced by `[MASK]` with probability t and kept unchanged otherwise, producing the corrupted state x_t.

A bidirectional Transformer (Vaswani et al., 2017) p_θ is trained to reverse this process, predicting the clean token at each masked position conditioned on the corrupted state. With the noise level sampled uniformly, training minimizes the denoising objective

$$\mathcal{L}(\theta) = -\mathbb{E}_{x_0,\, t,\, x_t} \left[\frac{1}{t} \sum_{i=1}^{L} \mathbf{1} \left[x_t^i = \text{[MASK]} \right] \log p_\theta \left(x_0^i \mid x_t \right) \right], \tag{1}$$

which upper-bounds the negative log-likelihood of the data. At inference time, generation starts from a fully masked sequence and unmasks tokens through iterative denoising steps.

2.2 Mixture-of-Experts Transformers and Scaling Laws

A Mixture-of-Experts (MoE) Transformer (Jacobs et al., 1991; Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Zoph et al., 2022; Dai et al., 2024) decouples model capacity from per-token computation by replacing the feed-forward network with n_e routed experts and a lightweight router: the total parameter count scales with n_e, whereas each token is processed by only a small subset. The router activates the top-n_a routed experts per token and combines their outputs. Every token is also processed by a single shared expert with intermediate width $d_{\text{share}} = n_s d_{\text{expert}}$, where n_s denotes shared capacity in routed-expert units. Following prior MoE scaling studies (Clark et al.,

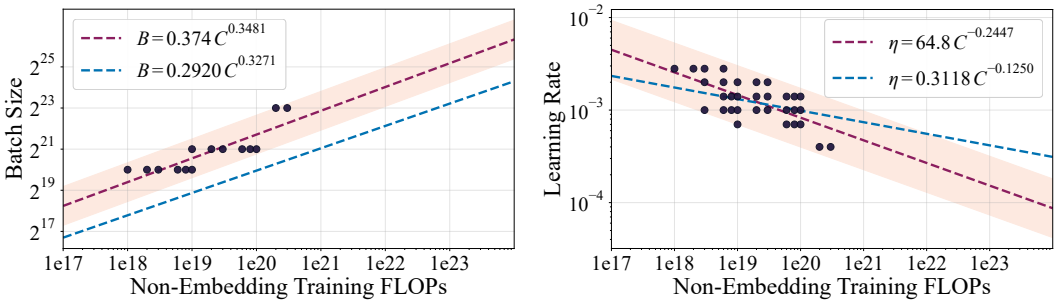


Figure 1: **Scaling curves of token batch size and learning rate with training compute. Left:** token batch size. **Right:** learning rate. Magenta dashed lines indicate our fitted scaling laws, blue dashed lines show the reference scaling laws from DeepSeek LLM (Bi et al., 2024), and shaded regions denote the empirical ranges around our fitted curves.

2022; Krajewski et al., 2024; Abnar et al., 2025; Ludziejewski et al., 2025; Tian et al., 2026), we parameterize three key architectural choices: the activation ratio $A = (n_a + n_s)/(n_e + n_s)$, the fraction of expert capacity activated per token; the expert granularity $G = 2d_{\text{model}}/d_{\text{expert}}$, controlling how routed capacity is partitioned into experts; and the shared-expert ratio $S = n_s/(n_a + n_s)$, controlling the fraction of activated expert capacity assigned to the shared pathway.

Scaling laws use small-scale findings to estimate how compute-dependent choices, including optimization hyperparameters, model–data allocation, and architecture, should change at larger budgets (Hestness et al., 2017; Kaplan et al., 2020; Hoffmann et al., 2022; Krajewski et al., 2024; Abnar et al., 2025). In practice, power-law fits model the optimal batch size and learning rate as functions of the compute budget C, while IsoFLOP analysis fixes C and sweeps model-side computation against training tokens (Hoffmann et al., 2022; Li et al., 2025). Dense-model studies approximate training compute as $C \approx 6ND$, with N non-embedding parameters and D training tokens, assuming all parameters process every token (Kaplan et al., 2020; Hoffmann et al., 2022). Sparse activation breaks this assumption, so we follow MoE scaling studies (Bi et al., 2024; Ludziejewski et al., 2025) and use $C = MD$, where M denotes the activated non-embedding training FLOPs per token, counted in forward and backward passes; full compute expressions are given in Appendix A.1.

3 Scaling Laws for MoE dLLMs

In this section, we develop scaling laws and design principles for MoE dLLMs in three stages. We first calibrate compute-dependent batch size and learning rate, then estimate the optimal allocation between activated model-side computation and training data, and finally derive principles for decomposing this budget into activation ratio, expert granularity, and shared-expert ratio.

3.1 Scaling Law for Hyperparameters

Hyperparameter scaling is well studied for AR models (Bi et al., 2024; Li et al., 2025), but whether its compute-dependent laws describe dLLMs is unclear: under the masked denoising objective, each update supervises only sampled masked positions, so the token batch size does not correspond to the effective number of prediction targets—under a uniform timestep schedule, only about half of the tokens are predicted in expectation. This reduced supervision can shift gradient noise and learning-rate sensitivity, moving the optimal batch size and learning rate at a given compute budget. We therefore study how they vary with compute, both to recalibrate AR recipes for dLLM training and to establish stable, compute-efficient hyperparameter settings for subsequent analyses.

Concretely, we perform a hyperparameter grid search over token batch size and learning rate across a set of representative model scales, ranging from 158M to 3.6B, under compute budgets from 10^{18} to 3×10^{20}; the detailed training and model settings are provided in Appendix A.3. For each compute budget C, we identify the optimal batch size and learning rate from the final training loss and regress them on C with linear fits in log–log space, as shown in Figure 1. The resulting scaling laws can be

summarized as

$$B^* = 0.374 \cdot C^{0.3481}, \quad \eta^* = 64.8 \cdot C^{-0.2447}, \tag{2}$$

where B^* and η^* denote the optimal token batch size and learning rate, respectively.

To test whether the fitted dLLM-specific scaling law extrapolates beyond fitting range, we continue the joint batch-size and learning-rate runs from 3×10^{20} to 6×10^{20} FLOPs and examine the loss grid in Figure 2. The best observed configuration nearly coincides with the fitted recommendation, and no neighboring configuration attains a lower loss: the fitted law remains predictive at twice its largest fitting budget, supporting its use as an estimate of compute-dependent hyperparameters beyond the fitting range.

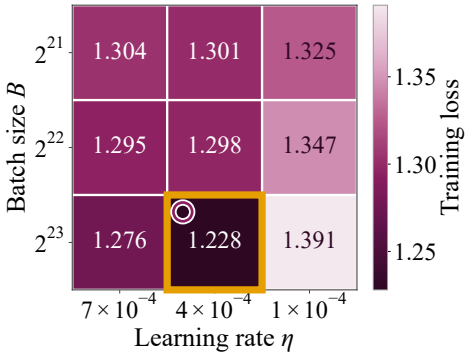


Figure 2: **Joint search validation over batch size and learning rate at 6×10^{20} FLOPs.** Each cell corresponds to one training run, with color and overlaid value denoting training loss; the magenta ring marks the fitted scaling-law prediction and the gold frame the best observed configuration.

Optimization: the optimal batch size is larger, and the optimal learning rate decays faster than AR laws predict. The optimization directions match AR training: the optimal batch size grows sublinearly with compute and the optimal learning rate decreases at larger compute. The calibration, however, differs: compared with DeepSeek LLM's AR law (Bi et al., 2024), our dLLM fit exhibits a slightly steeper batch-size scaling and a faster decline of the optimal learning rate with compute, shifting the high-compute estimate toward larger batches and smaller learning rates. At 10^{20} FLOPs, for example, DeepSeek LLM's AR law predicts a similar learning rate, 9.86×10^{-4} compared with 8.27×10^{-4} from our fit, but a much smaller optimal batch size, 1.02M tokens compared with 3.43M tokens. This batch-size gap is consistent with the reduced effective supervision per token noted above. Thus, AR scaling can provide a useful prior, but it should not replace dLLM-specific calibration when the fitted trends diverge. All subsequent experiments therefore use our fitted dLLM hyperparameter laws.

3.2 Scaling Law for Compute Allocation

With hyperparameters calibrated by these laws, the next issue is how to divide a fixed compute budget between activated model-side computation and training data. For dLLMs, supervising only masked tokens yields fewer prediction targets per token than AR training, so the AR balance may not hold. IsoFLOP studies characterize this trade-off for AR models, both dense (Hoffmann et al., 2022; Bi et al., 2024) and MoE (Ludziejewski et al., 2025), and for dense dLLMs (Nie et al., 2025; Ni et al., 2025b; von Rütte et al., 2026); MoE dLLM allocation remains uncharacterized.

To address this gap, we measure the model side by activated non-embedding FLOPs per token M and the data side by training tokens D, giving $C = MD$. We sweep their allocation across fixed compute budgets from 10^{17} to 10^{20} training FLOPs, with detailed settings in Appendix A.4.

The left panel of Figure 3 shows the resulting IsoFLOP curves: at each budget, the loss is U-shaped in the allocation, with smaller models capacity-limited despite seeing more tokens and larger models data-limited as fewer tokens fit the same budget. The lowest-loss point of each curve, marked by a star, gives the empirical compute-optimal allocation (M^*, D^*). As the budget increases, these optima move toward larger M and D, tracing the model-side and data-side frontiers $M^*(C)$ and $D^*(C)$. The middle and right panels show that both frontiers are approximately linear in log–log space. Fitting them under the fixed-compute constraint gives

$$M^*(C) = 0.5152 \cdot C^{0.475}, \quad D^*(C) = 1.9411 \cdot C^{0.525}. \tag{3}$$

Compute allocation: the optimal allocation between model and data is nearly balanced, with a slight tilt toward data. Both frontiers grow with compute at near-balanced rates: the data-side exponent of 0.525 slightly exceeds the model-side exponent of 0.475, so the optimal token budget grows faster than activated model-side computation. Table 2 places this tilt alongside representative compute-allocation laws for AR models and dLLMs.

Table 2: **Representative compute-allocation scaling laws for language models.** We position our MoE dLLM frontier against representative model–data allocation laws across modeling objectives and architectures. The frontier columns report fitted growth exponents. For DLMs, we report the masked setting matching our formulation; the SMDM coefficients are taken from the DLMs analysis.

Scaling law	Modeling	Architecture	Model frontier	Data frontier
Kaplan (Kaplan et al., 2020)	AR	Dense	$M^* \propto C^{0.73}$	$D^* \propto C^{0.27}$
Chinchilla (Hoffmann et al., 2022)	AR	Dense	$M^* \propto C^{0.49}$	$D^* \propto C^{0.51}$
DeepSeek LLM (Bi et al., 2024)	AR	Dense	$M^* \propto C^{0.5243}$	$D^* \propto C^{0.4757}$
Llama 3 (Grattafiori et al., 2024)	AR	Dense	$M^* \propto C^{0.463}$	$D^* \propto C^{0.537}$
SMDM (Nie et al., 2025)	AR	Dense	$M^* \propto C^{0.644}$	$D^* \propto C^{0.356}$
Ling (Tian et al., 2026)	AR	Dense	$M^* \propto C^{0.5422}$	$D^* \propto C^{0.4578}$
Ling (Tian et al., 2026)	AR	MoE	$M^* \propto C^{0.5095}$	$D^* \propto C^{0.4905}$
SMDM (Nie et al., 2025)	Diffusion	Dense	$M^* \propto C^{0.634}$	$D^* \propto C^{0.366}$
Quokka (Ni et al., 2025b)	Diffusion	Dense	$M^* \propto C^{0.514}$	$D^* \propto C^{0.486}$
DLMs (von Rütte et al., 2026)	Diffusion	Dense	$M^* \propto C^{0.566}$	$D^* \propto C^{0.434}$
Ours	Diffusion	MoE	$M^* \propto C^{0.475}$	$D^* \propto C^{0.525}$

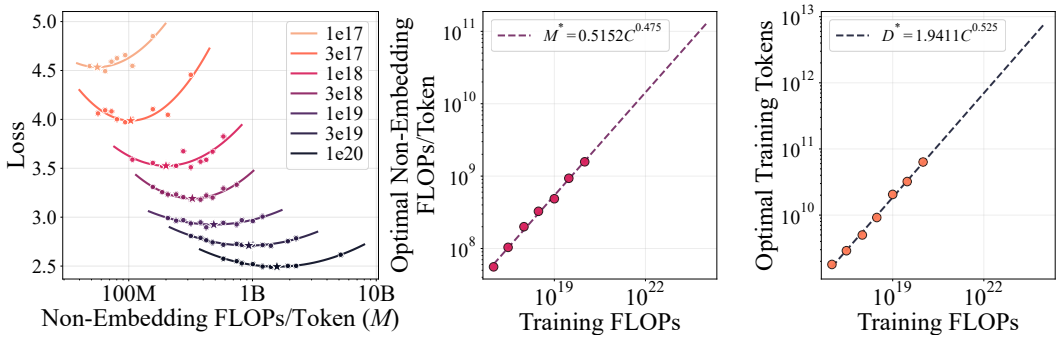


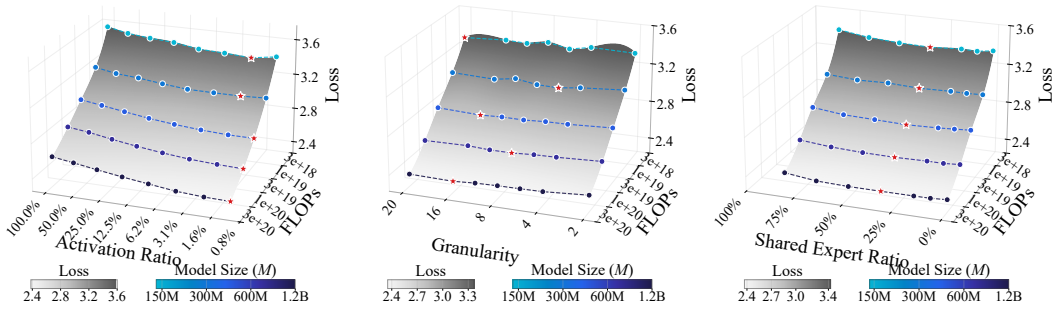
Figure 3: **IsoFLOP analysis for compute allocation. Left**: training loss under different model–data allocations at fixed compute budgets, with stars marking the lowest-loss. **Middle and right**: fitted power laws for the compute-optimal non-embedding FLOPs per token and training tokens.

Within this comparison, two matched results separate the effects of architecture and modeling objective. First, Ling compares dense and MoE architectures under AR modeling: moving from dense to MoE shifts the model/data exponents from $0.5422/0.4578$ to $0.5095/0.4905$, indicating that sparse activation favors data. Second, SMDM compares AR and dLLM objectives under a dense architecture: the exponents shift from $0.644/0.356$ to $0.634/0.366$, in the same direction. Our setting combines both departures, and among dLLM frontiers in Table 2, ours is the only one tipping toward the data side (0.525)—consistent with these two tendencies acting jointly. This tilt echoes evidence that dLLMs are stronger data learners than AR models (Ni et al., 2025a; Prabhudesai et al., 2026).

The fitted frontier yields a simple allocation rule: marginal compute is better spent on more training tokens than on activated FLOPs per token. Following it fixes the activated model-side budget $M^*(C)$ at each scale while leaving expert count, expert size, and activation pattern undetermined.

3.3 Scaling Law for MoE Architecture

The allocation law fixes the activated computation per token, but not how this computation is divided among experts. This structuring choice poses a dLLM-specific problem: existing MoE design guidelines come from AR training (Krajewski et al., 2024; Dai et al., 2024), while the dLLM router acts on masked denoising states that vary with noise level and mask pattern. We therefore re-examine the three architectural dimensions defined in Section 2: the activation ratio A, which controls routing sparsity and thereby the size of the routed expert pool; the expert granularity G, which trades off routing diversity against per-expert capacity; and the shared-expert ratio S, which determines the



(a) Activation Ratio (A) **(b)** Expert Granularity (G) **(c)** Shared-Expert Ratio (S)

Figure 4: **MoE architecture scaling laws.** We evaluate how different decompositions of the activated model-side budget affect training loss across compute scales. **(a):** Activation Ratio A, **(b):** Expert Granularity G, and **(c):** Shared-Expert Ratio S. Colors indicate the activated model-side budget $M^*(C)$, and red stars mark the lowest-loss configurations at each compute scale.

fraction of activated capacity assigned to a shared pathway. We ask how each dimension should optimally be configured as the training compute budget grows by multiple orders of magnitude.

Concretely, at five reference budgets C from 6×10^{17} to 6×10^{19} FLOPs, we fix the activated model-side budget to $M^*(C)$ and sweep one dimension at a time while holding the other two fixed. Each candidate is trained for $3D^*(C)$ tokens, placing the sweeps in an overtraining regime representative of large-scale pretraining (Grattafiori et al., 2024; Yang et al., 2025; Tian et al., 2026), with batch size and learning rate from the fitted hyperparameter laws at the resulting training compute of approximately $3C$. For example, at $C = 2 \times 10^{19}$, each candidate is sized by $M^*(2 \times 10^{19})$, trained for $3D^*(2 \times 10^{19})$ tokens, and assigned hyperparameters from the fitted laws at 6×10^{19} FLOPs. Detailed settings are in Appendix A.5; Figure 4 summarizes the training results across these sweeps.

Activation ratio: larger compute budgets favor sparser activation. We examine the activation ratio A in Figure 4(a). At fixed $M^*(C)$, reducing A generally lowers training loss, with the benefit growing as compute increases. At the smallest budgets, the second-lowest activation ratio slightly outperforms the sparsest setting. This exception is consistent with the intuition that extremely sparse routing creates a larger routed expert pool that requires sufficient compute to exploit; the observed trend suggests that MoE dLLMs increasingly benefit from this capacity as the budget grows.

Expert granularity: $G = 8$–16 remains robust across scales. Figure 4(b) presents the same sweep for granularity G. G does not show a monotonic trend across compute scales, suggesting that it is not a primary scaling direction. Instead, it mainly reflects a trade-off between routing diversity and per-expert capacity—coarser experts provide stronger individual transformations but fewer routing choices, while finer-grained experts increase routing choices at the cost of narrower experts. For MoE dLLMs, this trade-off may be more pronounced than in AR training, as the router acts on corrupted states that vary with noise level and mask pattern, where diverse specialization and sufficient per-expert expressiveness matter. Within this trade-off, $G = 8$ to $G = 16$ remains near-optimal across the studied scales.

Shared-expert ratio: $S = 33.3\%$ remains optimal across scales. The shared-expert ratio S, shown in Figure 4(c), behaves differently. Across all compute scales, the loss curves are U-shaped and reach their minima at $S = 33.3\%$. This optimum contrasts with AR MoE designs: DeepSeek-MoE adopts $S = 25\%$, Qwen3 uses no shared experts (Dai et al., 2024; Yang et al., 2025), and Tian et al. (2026) report a decreasing optimal ratio from 16.7% to 8.3%, motivating a fixed "one shared expert" rule. Our dLLM sweeps instead favor a shared pathway whose capacity grows in proportion to the activated model budget; a fixed shared component, as in the AR heuristic, would see its relative contribution diminish with scale. Our sweeps therefore support a dLLM-specific rule: maintain approximately one unit of shared activated capacity for every two units of routed activated capacity.

4 TRAINING LARGE-SCALE MOE DLLMS FROM SCRATCH

Guided by the scaling laws derived above, we design LLaDA MoE v2, a 30B-A3B MoE dLLM, and train it on 23.5T tokens. Each of the scaling components above contributes to the recipe: the allocation law guides the activated model size and token budget, the architecture principles determine key architectural choices (A, G, S), and the hyperparameter laws inform the batch size and learning rate at this scale. We briefly introduce its architecture and training below, with details in Appendix B.

Model design. LLaDA MoE v2 has 30B total parameters, with 3B parameters activated per token. The model–data split follows the allocation law with a deliberate data-side shift: the 23.5T-token budget is $4\times$ the extrapolated compute-optimal allocation $D^*(C)$ at this scale, extending the over-trained regime of the architecture sweeps in Section 3.3 to balance training compute against inference efficiency. For the MoE architecture, we adopt $(A, G, S) = (9.09\%, 8, 33.3\%)$, consistent with the sweep findings: sparser activation is favored at larger scales, $G = 8$ lies in the robust 8–16 range, and $S = 33.3\%$ is stable across scales. Each layer comprises 128 fine-grained routed experts with top-8 routing and one shared expert of width $4d_{\text{expert}}$, corresponding to $n_s = 4$.

Training strategy. We train LLaDA MoE v2 in five stages. Stages 1 and 2 each use 10T tokens, followed by 2T tokens of annealing in Stage 3. In Stage 4, we increase the RoPE base from 10,000 to 500,000, extend the context length from 4K to 32K, and continue training for 500B tokens. Stage 5 concludes with 1T tokens of long-context annealing, yielding the final pretrained base model. The pretraining learning rates and batch size fall within the empirical range of the fitted hyperparameter laws $B^*(C)$ and $\eta^*(C)$ in Figure 1, when extrapolated to this scale; details are in Appendix B.2.

4.1 BENCHMARK RESULTS

To assess whether the scaling-law-guided design translates into downstream performance at scale, we compare LLaDA MoE v2 30B-A3B with five representative dLLM and strong AR baselines in Table 3. The closest scale-matched baselines are SDAR Sci and Qwen3 30B-A3B: the former is obtained by continued pretraining from Qwen3 (Cheng et al., 2026a), whereas the latter is a strong AR MoE model (Yang et al., 2025). LLaDA MoE v2 is trained from scratch on 23.5T tokens, 63% of SDAR Sci's 37.05T and 65% of Qwen3's 36T. Evaluation settings are provided in Appendix B.4.

Across all benchmarks, LLaDA MoE v2 achieves the highest average among the evaluated dLLMs (58.60), outperforming SDAR Sci Base by 3.78 points and smaller dLLM baselines by at least 12.44 points, with its largest gains over SDAR Sci on coding benchmarks (HumanEval $+16.46$, BigCodeBench $+7.98$). Against Qwen3, it remains close on some coding benchmarks (OlympiadBench -2.22, MBPP -3.40), with gaps on the Chinese knowledge benchmarks CEval and CMMLU. These results make LLaDA MoE v2 the strongest dLLM trained from scratch to date: it leads every from-scratch baseline by a double-digit margin, and surpasses SDAR Sci Base, which builds on 36T tokens of AR pretraining. These scaling laws were fitted at budgets three orders of magnitude below this run, indicating that small-scale measurements can guide large-scale MoE dLLM training.

Figure 5 provides a more direct test of scaling-law guidance, comparing LLaDA MoE v2 30B-A3B checkpoints at varying training FLOPs against LLaDA MoE, a prior MoE dLLM developed without such guidance. Across diverse benchmarks spanning knowledge, mathematics, and code, checkpoints of LLaDA MoE v2 match or exceed LLaDA MoE at lower compute budgets. At approximately half of LLaDA MoE's training FLOPs, LLaDA MoE v2 surpasses it on MMLU, GSM8K, and KorBench; it does so on HellaSwag with less than 20% of the training FLOPs. This benchmark-versus-compute comparison provides practical evidence that scaling-law-guided design improves the efficiency with which pretraining compute is translated into downstream performance.

4.2 SUPERVISED FINE-TUNING

We further examine whether the benefits of scaling-law-guided pretraining transfer to downstream use through standard SFT. Starting from the LLaDA MoE v2 30B-A3B base checkpoint, we construct an instruct model and evaluate it on mathematical reasoning and code generation benchmarks.

We fine-tune the base model for three epochs on 7M instruction–response examples, primarily mathematical reasoning and code generation tasks. For each example, we keep the prompt uncorrupted, apply the masking process in Equation 1 only to the response, and compute the denoising loss over

Table 3: **Benchmark results.** We report results for LLaDA MoE v2, our 30B-A3B dLLM trained from scratch, alongside representative dLLMs (Cheng et al., 2026a; Zhu et al., 2025; Ye et al., 2025; Nie et al., 2026b) and the strong AR model Qwen3 (Yang et al., 2025). CPT denotes continued pretraining from AR models. Unmarked values are from our unified evaluation; * and † mark results from the original Qwen3 and LLaDA MoE papers, respectively. **Bold** and underline mark the best and second-best results among dLLMs; Qwen3 is shown for reference.

	LLaDA MoE v2	SDAR Sci	LLaDA MoE	Dream 7B	LLaDA 8B	Qwen3
Architecture	MoE	MoE	MoE	Dense	Dense	MoE
Modeling	Diffusion	Diffusion	Diffusion	Diffusion	Diffusion	AR
Method	Pretrain	CPT	Pretrain	CPT	Pretrain	Pretrain
# Total Params	30B	30B	7B	7B	8B	30B
# Activated Params	3B	3B	1B	7B	8B	3B
# Trained Tokens	23.5T	36 + 1.05T	21T	18 + 0.58T	2.3T	36T
General Tasks						
MMLU	78.01	**82.72**	64.59†	69.50†	65.90†	81.38*
MMLU-Pro	**57.28**	56.96	39.16†	48.15†	41.80†	61.49*
CEval	76.11	**86.95**	65.56†	59.18†	70.50†	87.50
CMMLU	77.99	**85.82**	65.65†	60.87†	69.90†	86.35
HellaSwag	**77.19**	56.38	65.46	74.37	70.82	77.92
KorBench	**45.92**	40.08	31.20†	37.44†	33.68†	44.96
Reasoning Tasks						
GSM8K	83.93	**86.13**	66.41†	77.79†	70.70†	91.81*
MATH	**54.72**	48.52	36.10†	39.60†	27.30†	59.04*
OlympiadBench	**28.74**	24.44	10.07†	10.22†	6.85†	30.96
Coding Tasks						
CRUXEval	50.62	**53.00**	38.94	40.31	36.38	56.88
MBPP	**71.00**	60.40	52.40†	56.20†	38.20†	74.40*
MultiPL-E	**53.78**	33.66	41.13†	27.60†	23.61†	66.53*
HumanEval	50.00	33.54	45.73†	**57.90**†	33.50†	52.44
LiveCodeBench v6	31.86	**39.87**	16.18†	14.87†	2.53†	49.18
BigCodeBench	**41.84**	33.86	21.23†	18.33†	13.42†	45.70
Average	**58.60**	54.82	43.99	46.16	40.34	64.44

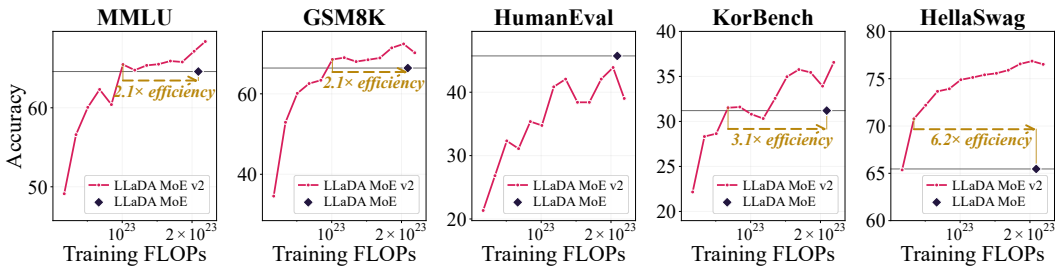


Figure 5: **Benchmark performance versus training compute.** Red curves show the scaling-law-guided LLaDA MoE v2 model checkpoints evaluated with varying training-FLOP budgets, and the black diamond denotes LLaDA MoE, trained without scaling-law guidance. Efficiency arrows show v2 surpasses LLaDA MoE scores with 2.1–6.2× fewer training FLOPs on multiple tasks.

the masked response positions, so that SFT optimizes the same objective as pretraining. We use a batch size of 512 sequences and a peak learning rate of 5.0×10^{-6} with linear warmup over the first 8% of steps; full optimization settings and data processing are given in Appendix B.3.

Table 4: **SFT results.** Comparison of the LLaDA MoE v2 Instruct with SDAR Chat on mathematics and code benchmarks. Values marked with † are from the SDAR paper.

	Reasoning				Code			
Model	Math	Olympiad	AIME24	AIME25	MBPP	LCB v6	BigCode	MultiPL-E
SDAR Chat	77.80†	34.93	16.70†	10.80†	71.60†	21.70†	**39.39**	45.00
LLaDA MoE v2	**80.02** (+2.22)	**46.44** (+11.51)	**30.00** (+13.30)	**20.00** (+9.20)	**81.03** (+9.43)	**27.75** (+6.05)	35.53 (-3.86)	**67.52** (+22.52)

We compare the instruct model with SDAR Chat 30B-A3B (Cheng et al., 2026a), a competitive MoE dLLM baseline. As Table 4 shows, despite being pretrained from scratch on fewer tokens rather than adapted from an AR checkpoint, LLaDA MoE v2 outperforms SDAR Chat on all four reasoning and three of four code benchmarks, with BigCodeBench as the only exception (-3.86). The gains are largest on competition-level reasoning and multi-language code: AIME 24/25 (AIME, 2025) ($+13.30$ and $+9.20$), OlympiadBench ($+11.51$), and MultiPL-E ($+22.52$). These results confirm that the benefits of scaling-law-guided pretraining persist through standard SFT.

5 Related Work

Diffusion Language Models (dLLMs) have emerged as a new paradigm for language modeling (Austin et al., 2021a; Campbell et al., 2022; Lou et al., 2023; Shi et al., 2024; Ou et al., 2025; Sahoo et al., 2024; Zheng et al., 2025). dLLMs model the data distribution over complete sequences through iterative masked denoising with bidirectional context, giving rise to training dynamics distinct from those of AR training. Recent work has scaled dLLMs both by pretraining from scratch (Nie et al., 2026b;a; Zhu et al., 2026; 2025) and by adapting pretrained AR models (Gong et al., 2025; Ye et al., 2025; Cheng et al., 2026a; Bie et al., 2025; Team et al., 2026a). Meanwhile, several studies have begun to characterize the scaling behavior of dLLMs (Nie et al., 2025; Ni et al., 2025b; von Rütte et al., 2026; Sahoo et al., 2026), though most of them are for dense architectures, with OpenMoE 2 (Ni et al., 2025c) an exception exploring some MoE designs for dLLMs. Its findings, however, probe isolated choices over few budgets, without jointly analyzing optimization, compute allocation, and architecture or validating at scale; we instead derive compute-dependent laws from systematic multi-budget sweeps and validate them at the 30B scale.

Mixture-of-Experts (MoE) and Scaling Laws. By replacing the feed-forward network with fine-grained experts and activating a subset of parameters per token, MoE architectures increase capacity without proportional extra computation (Jacobs et al., 1991; Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Zoph et al., 2022; Du et al., 2022; Dai et al., 2024). Building on language-model scaling laws (Hestness et al., 2017; Kaplan et al., 2020; Hoffmann et al., 2022; Bi et al., 2024; Li et al., 2025), recent work applies scaling insights to compute-efficient MoE design (Clark et al., 2022; Krajewski et al., 2024; Abnar et al., 2025; Ludziejewski et al., 2025; Tian et al., 2026), and several works apply MoE to dLLMs (Zhu et al., 2025; Cheng et al., 2026a; Feng et al., 2026; Zhang et al., 2026). Existing MoE dLLMs, however, largely inherit AR designs, leaving their optimization, compute allocation, and expert-architecture scaling behavior largely uncharacterized.

6 Conclusion

We ask whether AR-inherited training recipes transfer to MoE dLLMs and answer through a systematic scaling study: across optimization hyperparameters, compute allocation, and expert architecture, AR trends remain useful priors but require dLLM-specific calibration. The resulting laws and design principles guide the pretraining of LLaDA MoE v2 30B-A3B on 23.5T tokens, three orders of magnitude beyond the fitting budgets. LLaDA MoE v2 also surpasses LLaDA MoE with only half the latter's training FLOPs. Its average score exceeds SDAR Sci 30B-A3B Base by 3.78 points and approaches Qwen3 30B-A3B with fewer pretraining tokens. After SFT, it outperforms SDAR Chat 30B-A3B on seven of eight mathematics and coding tasks, making it the strongest from-scratch dLLM to date. Although we do not fully explore the interactions among all design choices, we expect these laws and principles to guide the training and application of MoE dLLMs.

AI USE STATEMENT

In this work, we used generative AI tools, including large language models, to understand, read, and edit software code for training and evaluation, and to check the manuscript for grammar and readability and perform preliminary language polishing. Additionally, we used generative AI tools to search for and identify related work. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate or prove mathematical claims, design experiments, or interpret results; the remaining disclosure categories are not applicable to this work. All AI-assisted code was reviewed, tested, and verified for correctness by the authors, and all AI-assisted text was checked against the original experimental records. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work characterizes the scaling behavior of Mixture-of-Experts diffusion language models and uses the resulting scaling laws to guide the from-scratch training of a 30B-A3B model. However, the web-scale text corpora used in our scaling-law exploration and large-scale pretraining cannot be exhaustively verified by humans, and may still contain discriminatory, biased, toxic, or otherwise harmful content that the resulting models could reproduce. To mitigate these risks, we filtered the training corpora to remove harmful content, and we additionally mixed data aligned with human preferences into the supervised fine-tuning stage to reduce the probability that the model generates harmful outputs. Despite these measures, the model may still generate harmful content, and we will continue to monitor its outputs to support a responsible release of the model.

REPRODUCIBILITY STATEMENT

This work develops scaling laws for the optimization hyperparameters, model–data allocation, and expert architecture of MoE dLLMs, and applies them to train LLaDA MoE v2 30B-A3B from scratch. The settings needed to reproduce the scaling-law experiments are described in Section 3, with the full model configurations, search grids, and training hyperparameters provided in Appendices A.3–A.5, and the compute accounting and MoE implementation details in Appendices A.1 and A.2. The architecture of LLaDA MoE v2 is fully specified in Section 4 and Appendix B.1, and its pretraining, fine-tuning, and evaluation protocols are described in Appendices B.2–B.4. We will open-source LLaDA MoE v2 Base and LLaDA MoE v2 Instruct to help the community reproduce our evaluation results as closely as possible.

REFERENCES

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin Mohamed Elnouby Ali, Josh Susskind, and Vimal Thilak. Parameters vs flops: Scaling laws for optimal sparsity for mixture-of-experts language models. *arXiv preprint arXiv:2501.12370*, 2025.

AIME. AIME problems and solutions, 2025. URL `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 4895–4901, 2023.

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.

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, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

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.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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, 2026a.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026b.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International conference on machine learning*, pp. 4057–4086. PMLR, 2022.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1280–1297, 2024.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Sicheng Feng, Zigeng Chen, Gongfan Fang, Xinyin Ma, and Xinchao Wang. dmoe: dllms with learnable block experts. *arXiv preprint arXiv:2605.30876*, 2026.

Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. In *International Conference on Learning Representations*, volume 2025, pp. 67661–67682, 2025.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations*, volume 2025, pp. 5046–5073, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

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.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in neural information processing systems*, 36:62991–63010, 2023.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

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.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11260–11285, 2024.

Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, et al. Predictable scale: Part i, step law–optimal hyperparameter scaling law in large language model pretraining. *arXiv preprint arXiv:2503.04715*, 2025.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, et al. Joint moe scaling laws: Mixture of experts can be memory efficient. *arXiv preprint arXiv:2502.05172*, 2025.

Kaijing Ma, Xeron Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, et al. Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. In *International Conference on Learning Representations*, volume 2025, pp. 80062–80161, 2025.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2025, pp. 62061–62121, 2025.

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.

Jinjie Ni et al. Openmoe 2: Sparse diffusion language models. `https://github.com/JinjieNi/OpenMoE2`, 2025c.

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, Qiyang Min, Shaoxuan Xu, Zihao Huang, Yuxuan Song, Yong Shan, Yankai Lin, Wayne Xin Zhao, Chongxuan Li, and Ji-Rong Wen. Improved large language diffusion models. *arXiv preprint arXiv:2606.25331*, 2026a.

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, 2026b.

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.

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.

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 Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models. *arXiv preprint arXiv:2602.15014*, 2026.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

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

Yangyi Shen, Tianjian Feng, Jiaqi Han, Wen Wang, Tianlang Chen, Chunhua Shen, Jure Leskovec, and Stefano Ermon. Improving diffusion language model decoding through joint search in generation order and token space. *arXiv preprint arXiv:2601.20339*, 2026.

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.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, et al. Diffusiongemma technical report. *arXiv preprint arXiv:2608.00146*, 2026a.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026b.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2026, pp. 29806–29843, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4643–4663, 2024.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. *Advances in Neural Information Processing Systems*, 38:138867–138907, 2026.

Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Jiang, Zhenguo Li, Wei Bi, et al. Diffusion of thought: Chain-of-thought reasoning in diffusion language models. *Advances in Neural Information Processing Systems*, 37:105345–105374, 2024.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b, 2025. URL https://hkunlp.github.io/blog/2025/dream.

Zebin You, Shen Nie, Xiaolu Zhang, JUN ZHOU, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. Llada-v: Large language diffusion models with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10093–10105, 2026a.

Zebin You, Xiaolu Zhang, Jun Zhou, Chongxuan Li, and Ji-Rong Wen. Llada-o: An effective and length-adaptive omni diffusion model. *arXiv preprint arXiv:2603.01068*, 2026b.

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.

Shuibai Zhang, Caspian Zhuang, Chihan Cui, Zhihan Yang, Fred Zhangzhi Peng, Yanxin Zhang, Haoyue Bai, Zack Jia, Yang Zhou, Guanhua Chen, et al. Expert-choice routing enables adaptive computation in diffusion language models. *arXiv preprint arXiv:2604.01622*, 2026.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, et al. A survey of large language models. *Frontiers of Computer Science*, 20(12):2012627, 2026.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. In *International Conference on Learning Representations*, volume 2025, pp. 63186–63227, 2025.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, et al. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*, 2025.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11425–11460, 2026.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A SCALING LAWS FOR MOE DLLMS

A.1 COMPUTE ACCOUNTING

We distinguish total parameter count, activated parameter count, and the activated computation used in our scaling analysis. Let n_{layer} be the number of Transformer layers, d_{model} the hidden size, s the sequence length, and $r_{\text{kv}} = n_{\text{kvheads}}/n_{\text{heads}}$ the ratio of key-value heads to query heads. Each MoE layer contains n_e routed experts, each with intermediate width d_{expert}, of which n_a are selected per token. In the implemented architecture, every token is additionally processed by a single shared expert with intermediate width d_{share}. To express routed and shared capacity in common units, we define $n_s \equiv d_{\text{share}}/d_{\text{expert}}$, or equivalently $d_{\text{share}} = n_s d_{\text{expert}}$. Thus, n_s denotes the shared capacity measured in units of one routed expert, rather than the physical number of shared experts. With SwiGLU (Shazeer, 2020), the three projection matrices contain $3d_{\text{model}}d_{\text{expert}}$ parameters for each routed expert and $3d_{\text{model}}d_{\text{share}} = 3n_s d_{\text{model}}d_{\text{expert}}$ parameters for the single shared expert.

Ignoring biases and normalization parameters, the total and activated non-embedding parameter counts are

$$P_{\text{nonemb}} = n_{\text{layer}}\left[2d_{\text{model}}^2(1+r_{\text{kv}}) + d_{\text{model}}n_e + 3d_{\text{model}}(n_e d_{\text{expert}} + d_{\text{share}})\right], \quad (4)$$

$$P_{\text{act,nonemb}} = n_{\text{layer}}\left[2d_{\text{model}}^2(1+r_{\text{kv}}) + d_{\text{model}}n_e + 3d_{\text{model}}(n_a d_{\text{expert}} + d_{\text{share}})\right]. \quad (5)$$

The first term accounts for the query, key, value, and output projections; the second accounts for the router, which scores all n_e routed experts for every token; and the final term accounts for the routed experts and the single shared expert. Substituting $d_{\text{share}} = n_s d_{\text{expert}}$ recovers the equivalent-unit forms $3d_{\text{model}}d_{\text{expert}}(n_e + n_s)$ and $3d_{\text{model}}d_{\text{expert}}(n_a + n_s)$, respectively. We use an input embedding matrix and a separate LM head matrix, each containing Vd_{model} parameters for vocabulary size V. Their combined $2Vd_{\text{model}}$ parameters are included when reporting total or activated parameter counts but excluded from the non-embedding quantities used below.

We count one multiply–accumulate as two FLOPs. The forward FLOPs per token of one layer are approximated by

$$F_{\text{attn}} = 4d_{\text{model}}^2(1+r_{\text{kv}}) + 4sd_{\text{model}}, \quad (6)$$

$$F_{\text{MoE}} = 2d_{\text{model}}n_e + 6d_{\text{model}}(n_a d_{\text{expert}} + d_{\text{share}}), \quad (7)$$

where the two terms in F_{attn} correspond to the attention projections and the two sequence-level attention matrix multiplications, respectively. The terms in F_{MoE} account for routing, the selected routed experts, and the single shared expert. Approximating the backward pass as twice the forward pass, we define the activated non-embedding FLOPs per token as

$$M = 3n_{\text{layer}}\left[4d_{\text{model}}^2(1+r_{\text{kv}}) + 4sd_{\text{model}} + 2d_{\text{model}}n_e + 6d_{\text{model}}(n_a d_{\text{expert}} + d_{\text{share}})\right]. \quad (8)$$

This FLOPs accounting omits the input embedding, LM head, normalization operations, nonlinearities, and attention softmax. All Transformer layers in our models share the same architecture; accordingly, Equation 8 multiplies the per-layer FLOPs by n_{layer}.

Finally, let D denote the total number of tokens processed during training, including both masked and visible positions. The compute budget used throughout our scaling law experiments is

$$C = MD. \quad (9)$$

Masked and visible tokens incur the same Transformer computation, so the sampled corruption level changes the number of supervised prediction targets but not the accounted FLOPs.

A.2 MOE IMPLEMENTATION

Throughout the scaling experiments, we use a MoE Transformer architecture in which every layer pairs grouped-query attention (GQA) (Ainslie et al., 2023) with an MoE feed-forward block comprising the routed experts and the single shared expert parameterized in Appendix A.1, all implemented as SwiGLU networks; the only exceptions arise in the architecture experiments, where some configurations omit the shared expert.

Routing is performed independently for each token at every layer. Given the token representation h, a linear router produces logits $r(h) \in \mathbb{R}^{n_e}$ and routing scores $p(h) = \text{softmax}(r(h))$. Let $\mathcal{T}(h)$

contain the indices of the n_a largest routing scores. We renormalize the selected scores as

$$w_i(h) = \frac{p_i(h)}{\sum_{j \in \mathcal{T}(h)} p_j(h)}, \qquad i \in \mathcal{T}(h). \tag{10}$$

The routed expert output is

$$E_{\text{route}}(h) = \sum_{i \in \mathcal{T}(h)} w_i(h) E_i(h). \tag{11}$$

The routed and shared pathways are combined as $E_{\text{share}}(h) + \lambda E_{\text{route}}(h)$, where E_i is a routed expert, E_{share} is the shared expert, and λ balances the output scales of the two pathways. The routed-only configuration in the shared-expert-ratio experiments omits the shared term and does not use a scaling factor.

For configurations with $n_s > 0$, we estimate λ by matching the expected output norms of the shared and routed pathways at initialization, following the gate scaling heuristic of Liu et al. (2025). Treating the shared expert of width $n_s d_{\text{expert}}$ as n_s expert-width units, and assuming that all expert outputs have equal norms and are pairwise orthogonal at initialization, the shared pathway has norm proportional to $\sqrt{n_s}$, whereas the unscaled routed pathway has norm proportional to $(\sum_{i \in \mathcal{T}(h)} w_i(h)^2)^{1/2}$. Approximating the initialization distribution of the router logits by $r(h) \sim \mathcal{N}(0, I_{n_e})$, we estimate

$$\lambda = \mathbb{E}_{r(h) \sim \mathcal{N}(0, I_{n_e})} \left[\frac{\sqrt{n_s}}{\left(\sum_{i \in \mathcal{T}(h)} w_i(h)^2 \right)^{1/2}} \right], \tag{12}$$

where the expectation is approximated by Monte Carlo sampling independently for each architecture configuration with a shared expert.

We train the router jointly with the rest of the model. The training objective is

$$\mathcal{L} = \mathcal{L}_{\text{diff}} + \alpha_{\text{aux}} \mathcal{L}_{\text{bal}} + \alpha_z \mathcal{L}_z, \tag{13}$$

where $\mathcal{L}_{\text{diff}}$ is the denoising objective in Equation 1, $\mathcal{L}_{\text{bal}}$ is the load balancing auxiliary loss, and $\mathcal{L}_z$ is the router z-loss; following commonly used MoE training settings, we set $\alpha_{\text{aux}} = 0.01$ and $\alpha_z = 0.001$ (Fedus et al., 2022; Shazeer et al., 2017; Zoph et al., 2022). Both auxiliary terms are added directly to the denoising loss, and their coefficients are held fixed across all scaling and architecture sweeps.

A.3 Hyperparameter Scaling

We conduct the hyperparameter scaling experiments at three model scales—158M, 1B, and 3.6B—under compute budgets ranging from 10^{18} to 3×10^{20} FLOPs. All runs use the same pretraining data and a sequence length of 4096, and optimize the denoising objective in Equation 1. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and a weight decay of 0.1. The learning rate is linearly warmed up for 2,000 optimizer steps to the peak value η, allowing optimization to enter a stable regime before the peak rate is held constant until the final 10% of the training compute and then decayed to 0.1η using a cosine schedule.

For each model scale, we jointly search the global token batch size B and peak learning rate η, while holding the architecture and training-token budget fixed within each compute setting. Table 5 summarizes the model architectures, compute budgets, and corresponding search grids.

For each run, we define its loss as the average training loss over the final 0.5% of its allocated training FLOPs. At each compute budget, we identify the minimum average loss and regard configurations whose losses are no more than 0.25% above this minimum as near-optimal (Bi et al., 2024). We include all near-optimal configurations in log–log linear regressions of batch size and learning rate against C, yielding the final scaling laws.

A.4 Compute Allocation Scaling

We conduct IsoFLOP sweeps under compute budgets ranging from 10^{17} to 10^{20} FLOPs. All runs use the same pretraining data, denoising objective, and optimizer as the hyperparameter scaling

Table 5: **Configurations for the hyperparameter scaling sweep.** The upper block lists the model architectures; the lower block reports the compute budgets C (in FLOPs) and joint search grids for the global token batch size B and peak learning rate η.

Model architecture								
Model scale	n_{layer}	d_{model}	n_{heads}	n_{kvheads}	n_e	n_a	n_s	d_{expert}
158M	6	256	8	2	64	4	1	256
1B	10	640	10	2	64	4	1	640
3.6B	16	1024	16	4	64	4	1	1024

Search configuration			
Model scale	C	B	η
158M	$\{1, 2, 3, 6, 8\} \times 10^{18}$	$\{2^{17}, 2^{18}, 2^{19}, 2^{20}\}$	$\{1, 1.4, 2, 2.8\} \times 10^{-3}$
1B	$\{1, 2, 3, 6, 8, 10\} \times 10^{19}$	$\{2^{18}, 2^{19}, 2^{20}, 2^{21}\}$	$\{0.7, 1, 1.4, 2\} \times 10^{-3}$
3.6B	$\{0.8, 1, 2, 3\} \times 10^{20}$	$\{2^{21}, 2^{22}, 2^{23}\}$	$\{1, 4, 7\} \times 10^{-4}$

experiments in Appendix A.3. At each compute budget C, we set the token batch size and peak learning rate according to the fitted hyperparameter scaling laws reported in the main text. Tables 8 and 9 report the candidate model architectures and their corresponding allocation configurations, respectively. Because the number of optimizer steps varies across model–data allocations, we adapt the warmup length to each run. Let T denote the total number of optimizer steps implied by the allocation. We linearly warm up the learning rate to its peak over $T_{\text{warm}} = \max(0.01T, 100)$ steps, maintain the peak rate until the final 10% of the training steps, and then decay it to 10% of the peak using a cosine schedule. The 100-step floor provides a minimum warmup period for optimization stability (Nie et al., 2025).

For each compute budget, we evaluate a set of MoE models spanning different amounts of activated model-side computation. We measure the model side by the activated non-embedding FLOPs per token M, and assign each model a training-token budget $D = C/M$. At each compute budget, we select the evaluated allocation with the lowest loss as the empirical optimal allocation point. We fit the selected model-side optima using linear regression in log–log space and derive the corresponding data-side frontier from $D^*(C) = C/M^*(C)$, yielding the final compute-allocation scaling laws reported in the main text.

A.5 MoE Architecture Scaling

We conduct the MoE architecture sweeps at five reference compute budgets, $C \in \{6 \times 10^{17}, 2 \times 10^{18}, 6 \times 10^{18}, 2 \times 10^{19}, 6 \times 10^{19}\}$ FLOPs. At each reference budget, we keep the Transformer backbone fixed and vary one architectural dimension at a time while holding the other two fixed as closely as permitted by the discrete configurations. The sweep grids are parameterized using the definitions in Section 2: the activation ratio $A = (n_a + n_s)/(n_e + n_s)$, the expert granularity $G = 2d_{\text{model}}/d_{\text{expert}}$, and the shared-expert ratio $S = n_s/(n_a + n_s)$. We evaluate the fitted compute-allocation scaling laws at the reference budget C to obtain the target activated non-embedding FLOPs per token $M^*(C)$ and the compute-optimal token count $D^*(C)$. To make architecture selection representative of the overtraining regime commonly used in large-scale pretraining (Grattafiori et al., 2024; Gadre et al., 2025; Tian et al., 2026), we train each candidate for $3D^*(C)$ tokens, corresponding to approximately $3C$ training FLOPs. We therefore evaluate the fitted hyperparameter scaling laws at the actual training budget $3C$ and set the token batch size B and peak learning rate η accordingly. The common allocation targets and training hyperparameters are reported in Table 10; the exact model architectures used in the three sweeps are reported in Tables 11, 12, and 13. All remaining training settings are identical to those of the compute-allocation sweeps in Appendix A.4.

Within each sweep, the candidate architectures are constructed so that, ignoring the negligible router contribution, the activated model-side budget $M^*(C)$ is preserved. In the activation-ratio sweep, we fix the Transformer backbone, n_a, n_s, and d_{expert}, and vary only the number of routed experts n_e: the activated expert computation per token is unchanged, while the total parameter count grows as A

Table 6: **Architecture of LLaDA MoE v2 30B-A3B.**

Parameter scale		Transformer backbone				MoE configuration		
Total	Activated	n_{layer}	d_{model}	n_{heads}	n_{kvheads}	n_e	n_a	n_s
30.6B	3.4B	32	3072	32	4	128	8	4

decreases. In the expert-granularity sweep, we vary d_{expert} and scale n_e, n_a, and n_s in inverse proportion, preserving A, S, and both the routed and shared activated widths, $n_a d_{\text{expert}}$ and $n_s d_{\text{expert}}$. In the shared-expert-ratio sweep, we fix n_e and d_{expert} and redistribute the fixed activated expert width $(n_a + n_s)d_{\text{expert}}$ between the shared and routed pathways by varying n_s and n_a; since n_s enters the denominator of A, this redistribution induces a slight drift in the activation ratio across candidates, which is negligible and does not affect the controlled comparison.

For each run, we define its loss as the average training loss over the final 0.5% of its allocated training FLOPs, as in Appendix A.3. At each reference budget, we select the candidate with the lowest loss along each architectural dimension.

B Training Large-Scale MoE dLLMs

B.1 Model Architecture

We train LLaDA MoE v2 30B-A3B, a large MoE dLLM whose detailed architecture is reported in Table 6. The routing rule, the combination of routed and shared expert outputs, and the auxiliary objectives follow Appendix A.2.

The Transformer backbone contains 32 layers with hidden size 3072 and uses grouped-query attention (GQA) with 32 query heads and 4 key-value heads. The model uses a vocabulary of 157,184 tokens. All Transformer layers use an MoE feed-forward block.

To realize the architecture-scaling recommendations, we use $n_e = 128$ routed experts, activate $n_a = 8$ of them per token, and allocate $n_s = 4$ expert-width units to the single shared expert. Setting $G = 8$ gives $d_{\text{expert}} = d_{\text{model}}/4$ and hence $d_{\text{share}} = 4d_{\text{expert}} = d_{\text{model}}$. This discrete configuration yields $A = (8 + 4)/(128 + 4) = 9.09\%$ and $S = 4/(8 + 4) = 33.3\%$.

B.2 Pretraining

We pretrain LLaDA MoE v2 30B-A3B from scratch for a total of 23.5T tokens. The pretraining corpus is constructed from a broad collection of high quality text gathered from the web. We apply a standard data processing pipeline that collects the raw text, removes boilerplate and malformed or low quality documents, deduplicates repeated content, and filters harmful material. Throughout pretraining, we optimize the denoising objective in Equation 1; the routing procedure and auxiliary loss follow Appendix A.2.

The five-stage data schedule is summarized in Table 7. Stages 1 and 2 draw separate 10T-token samples from the same source corpus, with the Stage 2 mixture assigning slightly more weight to mathematical reasoning and code data. For Stage 3, we construct a curated 1T-token annealing corpus after cleaning, deduplication, and harmful-content filtering, and train on it for two epochs, yielding 2T training tokens. Stages 4 and 5 primarily use long-form data with sequence lengths of up to 32K tokens.

We train in BF16 precision using AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and a weight decay of 0.1 (Loshchilov & Hutter, 2017). The global token batch size is 33,554,432 tokens. Each stage uses a separate learning-rate schedule that linearly warms up from zero to its stage-specific peak over the first 2,000 optimizer steps. During the final 10% of the allocated training compute in Stages 1–4, the learning rate is decayed to the peak rate of the following stage using a cosine schedule; in Stage 5, it is decayed to 5.0×10^{-6} using the same schedule. The peak learning rates for Stages 1–5 are 1.5×10^{-4}, 1.0×10^{-4}, 5.0×10^{-5}, 1.0×10^{-5}, and 7.0×10^{-6}, respectively. At the transition from Stage 3 to Stage 4, we increase the RoPE base from 10,000 to 500,000 to extend the context

Table 7: **Five-stage pretraining schedule for LLaDA MoE v2 30B-A3B.**

Stage	Training phase	Tokens	Context length	RoPE base
1	Base pretraining 1	10T	4K	10,000
2	Base pretraining 2	10T	4K	10,000
3	Pretraining annealing	2T	4K	10,000
4	Context extension	500B	32K	500,000
5	Long-context annealing	1T	32K	500,000

length from 4K to 32K (Su et al., 2024; Xiong et al., 2024). The full pretraining run consumed approximately 460,000 NVIDIA B200 GPU hours.

B.3 Supervised Fine-Tuning

Starting from the final pretrained checkpoint, we fine-tune LLaDA MoE v2 30B-A3B for three epochs on 7M instruction–response examples, primarily comprising single-turn mathematical reasoning and code generation tasks. We process the data following the same general procedure used for pretraining, removing malformed or low quality examples, deduplicating repeated content, and filtering harmful material, and then format every example using a unified conversation template. The formatted examples are packed into non-overlapping 8K-token training sequences.

For each instruction–response pair, we concatenate the prompt and response but apply the forward masking process in Equation 1 only to response tokens, leaving the prompt available as uncorrupted conditioning context and computing the denoising loss only at masked response positions. The MoE load balancing loss and router z-loss described in Appendix A.2 remain active during SFT, with their coefficients unchanged at $\alpha_{\text{aux}} = 0.01$ and $\alpha_z = 0.001$.

We update all model parameters using AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, a weight decay of 0.1, and gradient clipping at a maximum norm of 1.0. The global batch size is 512 sequences. The learning rate is linearly warmed up to 5.0×10^{-6} over the first 8% of training steps and then follows a cosine schedule that decays it to a minimum of 1.0×10^{-6}. We use the final checkpoint as the instruct model.

To inspect inference-time routing, we use 16 prompts each from GSM8K, HumanEval, MATH, and MBPP, and generate the first 64-token block with the LLaDA MoE v2 Instruct. For each generated position, we record, in every MoE layer, the top-8 routed experts selected at the denoising step that decodes that position. Figure 6 reports how often each of the 128 routed experts in each layer appears in these recorded top-8 sets, measured as a percentage of the 16×64 generated positions, with darker cells indicating more frequent selection; the shared expert is omitted. Across tasks and layers, at least 104 of the 128 routed experts are selected at least once, preliminarily suggesting that LLaDA MoE v2 activates its experts broadly.

B.4 Evaluation

We evaluate LLaDA MoE v2 across a diverse suite of benchmarks covering general tasks (MMLU (Hendrycks et al., 2020), MMLU-Pro (Wang et al., 2024), CEval (Huang et al., 2023), CMMLU (Li et al., 2024), HellaSwag (Zellers et al., 2019), KorBench (Ma et al., 2025)), mathematical reasoning (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024)), and code generation (CRUXEval (Gu et al., 2024), MBPP (Austin et al., 2021b), MultiPL-E (Cassano et al., 2022), HumanEval (Chen et al., 2021), LiveCodeBench (Jain et al., 2025), BigCodeBench (Zhuo et al., 2025)).

For the base-model results in Table 3, we use conditional likelihood for multiple-choice tasks and conditional generation for the remaining tasks, whereas all results in the SFT evaluation in Table 4 are obtained through conditional generation. For each model, we preferentially report results from its official publication (Yang et al., 2025; Cheng et al., 2026a; Zhu et al., 2025; Ye et al., 2025; Nie et al., 2026b); when a benchmark result is unavailable, we report the score obtained under our unified evaluation configuration.

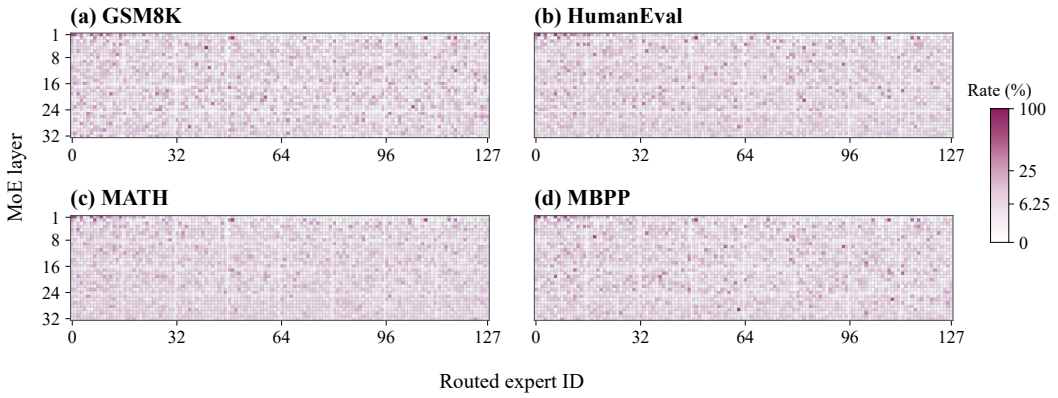


Figure 6: **Inference-time routed-expert selection across four task sets.** Rows index the 32 MoE layers and columns the 128 routed expert IDs. Each cell reports the percentage of generated positions whose recorded top-8 contains that expert.

We use conditional likelihood on MMLU, MMLU-Pro, CEval, CMMLU, and HellaSwag. For each example, we compute the conditional likelihood of every candidate answer given the prompt, select the candidate with the highest likelihood, and report accuracy over the benchmark. For an AR model, we compute the conditional log-likelihood under its left-to-right factorization. For LLaDA MoE v2, we follow the likelihood-evaluation protocols used in SMDM, LLaDA, and the previous LLaDA MoE (Nie et al., 2025; 2026b; Zhu et al., 2025). For SDAR, we estimate conditional likelihood using the method introduced in Block Diffusion (Arriola et al., 2025).

For conditional-generation tasks, each model generates a completion from the benchmark prompt using its native generation procedure. For code tasks, we extract the code from the response and execute it against the benchmark test cases. For mathematical reasoning tasks, we extract the final answer and determine correctness using the equivalence checker.

All instruct models are evaluated through conditional generation. We allow at most 1,024 generated tokens on each benchmark. Because this limit can be insufficient for models to produce a final answer on MATH, OlympiadBench, AIME 2024, and AIME 2025, we increase the limit to 4,096 tokens for these benchmarks. LLaDA MoE v2 uses semi-autoregressive sampling (Nie et al., 2026b) with a block size of 64 and a total number of denoising steps equal to the generation length. For SDAR, we follow its recommended block diffusion decoding procedure with a block size of 4 and likewise set the total number of sampling steps to the generation length (Cheng et al., 2026a).

Table 9: **Allocation configurations for the compute-allocation sweeps.** Each row reports the compute budget C (in FLOPs), model scale, training-token budget D, global token batch size B, and peak learning rate η.

C	Model scale	D ($\times 10^9$)	B	η
10^{17}	60M	2.07	2^{18}	4.4×10^{-3}
	63M	1.77		
	66M	1.55		
	69M	1.38		
	103M	1.25		
	111M	1.07		
	119M	0.934		
	184M	0.639		

Continued on next page

Table 9 continued

C	Model scale	D ($\times 10^9$)	B	η
3×10^{17}	63M	5.32	2^{19}	3.4×10^{-3}
	66M	4.65		
	69M	4.14		
	103M	3.74		
	111M	3.20		
	119M	2.80		
	184M	1.92		
	296M	1.45		
	472M	0.945		
10^{18}	119M	9.34	2^{20}	2.5×10^{-3}
	184M	6.39		
	242M	5.37		
	296M	4.83		
	325M	4.14		
	354M	3.62		
	472M	3.15		
	574M	2.63		
	575M	2.36		
	627M	2.10		
	731M	1.72		
3×10^{18}	184M	19.2	2^{20}	2.0×10^{-3}
	242M	16.1		
	296M	14.5		
	325M	12.4		
	354M	10.9		
	472M	9.45		
	574M	7.89		
	575M	7.09		
	627M	6.30		
	731M	5.16		
	1B	4.03		
10^{19}	296M	48.3	2^{21}	1.4×10^{-3}
	325M	41.4		
	354M	36.2		
	472M	31.5		
	574M	26.3		
	575M	23.6		
	627M	21.0		
	731M	17.2		
	1B	13.4		
	1.1B	12.2		
	1.4B	9.94		
	1.6B	8.29		

Continued on next page

Table 9 continued

C	Model scale	D ($\times 10^9$)	B	η
3×10^{19}	472M	94.5	2^{21}	1.1×10^{-3}
	574M	78.9		
	575M	70.9		
	627M	63.0		
	731M	51.6		
	1B	40.3		
	1.1B	36.6		
	1.4B	29.8		
	1.6B	24.9		
	1.8B	21.3		
	2.8B	15.5		
	3.2B	13.3		
10^{20}	731M	172	2^{22}	8.3×10^{-4}
	1B	134		
	1.1B	122		
	1.4B	99.4		
	1.6B	82.9		
	1.8B	71.0		
	2.8B	51.8		
	3.2B	44.4		
	7.5B	19.4		

Table 8: **Model architectures for the compute-allocation sweeps.** Each row specifies a candidate model architecture.

Model scale	n_{layer}	d_{model}	n_{heads}	n_{kvheads}	n_e	n_a	n_s	d_{expert}
60M	6	128	8	2	64	4	1	128
63M	7	128	8	2	64	4	1	128
66M	8	128	8	2	64	4	1	128
69M	9	128	8	2	64	4	1	128
103M	6	192	8	2	64	4	1	192
111M	7	192	8	2	64	4	1	192
119M	8	192	8	2	64	4	1	192
184M	8	256	8	2	64	4	1	256
242M	7	320	8	2	64	4	1	320
296M	6	384	8	2	64	4	1	384
325M	7	384	8	2	64	4	1	384
354M	8	384	8	2	64	4	1	384
472M	6	512	8	2	64	4	1	512
574M	6	576	8	2	64	4	1	576
575M	8	512	8	2	64	4	1	512
627M	9	512	8	2	64	4	1	512
731M	11	512	8	2	64	4	1	512
1B	10	640	10	2	64	4	1	640
1.1B	11	640	10	2	64	4	1	640
1.4B	10	768	12	4	64	4	1	768
1.6B	12	768	12	4	64	4	1	768
1.8B	14	768	12	4	64	4	1	768
2.8B	12	1024	16	4	64	4	1	1024
3.2B	14	1024	16	4	64	4	1	1024
7.5B	22	1280	20	4	64	4	1	1280

Table 10: **Common training configurations for the MoE architecture sweeps.** At each reference budget C, the allocation targets $M^*(C)$ and $D^*(C)$ are obtained from the fitted compute-allocation laws. All candidates are trained for $3D^*(C)$ tokens, corresponding to approximately $3C$ training FLOPs, while B and η are obtained by evaluating the fitted hyperparameter laws at $3C$.

C	$M^*(C)$	$3D^*(C)$	B	η
6×10^{17}	1.43×10^{8}	1.26×10^{10}	1048576	2.2×10^{-3}
2×10^{18}	2.54×10^{8}	2.36×10^{10}	1310720	1.7×10^{-3}
6×10^{18}	4.28×10^{8}	4.20×10^{10}	2097152	1.3×10^{-3}
2×10^{19}	7.59×10^{8}	7.91×10^{10}	3145728	9.4×10^{-4}
6×10^{19}	1.28×10^{9}	1.41×10^{11}	4194304	7.0×10^{-4}

Table 11: **Configurations for the activation-ratio sweep.** The upper block reports the backbone and fixed expert settings at each reference compute budget. The lower block lists the activation-ratio candidates shared across budgets, with each column giving one corresponding (n_e, A) pair.

Backbone and fixed expert settings							
Compute	n_{layer}	d_{model}	n_{heads}	n_{kvheads}	n_a	n_s	d_{expert}
6×10^{17}	8	256	8	2	2	1	256
2×10^{18}	8	448	8	2	2	1	448
6×10^{18}	10	512	16	4	2	1	512
2×10^{19}	12	640	16	4	2	1	640
6×10^{19}	15	768	16	4	2	1	768

Activation-ratio candidates								
Candidate	1	2	3	4	5	6	7	8
n_e	2	4	8	16	32	64	128	256
A (%)	100	60	33.3	17.6	9.1	4.6	2.3	1.2

Table 12: **Configurations for the expert-granularity sweep.** The upper block reports the Transformer backbone at each reference compute budget. In the lower block, each column gives one candidate's G, n_e, n_a, and n_s, together with the corresponding d_{expert} at every budget.

Transformer backbones				
Compute	n_{layer}	d_{model}	n_{heads}	n_{kvheads}
6×10^{17}	8	256	8	2
2×10^{18}	8	448	8	2
6×10^{18}	10	512	16	4
2×10^{19}	12	640	16	4
6×10^{19}	15	768	16	4

Expert-granularity candidates							
Candidate	1	2	3	4	5	6	7
G	2	4	6	8	12	16	20
n_e	64	128	192	256	384	512	640
n_a	2	4	6	8	12	16	20
n_s	1	2	3	4	6	8	10
$d_{\text{expert}}(6 \times 10^{17})$	256	128	85	64	43	32	26
$d_{\text{expert}}(2 \times 10^{18})$	448	224	149	112	75	56	45
$d_{\text{expert}}(6 \times 10^{18})$	512	256	171	128	85	64	51
$d_{\text{expert}}(2 \times 10^{19})$	640	320	213	160	107	80	64
$d_{\text{expert}}(6 \times 10^{19})$	768	384	256	192	128	96	77

Table 13: **Configurations for the shared-expert-ratio sweep.** The upper block reports the backbone and fixed expert settings at each reference compute budget. The lower block lists the shared-expert-ratio candidates shared across budgets, with each column giving one corresponding (n_a, n_s, S) tuple.

Backbone and fixed expert settings						
Compute	n_{layer}	d_{model}	n_{heads}	n_{kvheads}	n_e	d_{expert}
6×10^{17}	8	256	8	2	256	64
2×10^{18}	8	448	8	2	256	112
6×10^{18}	10	512	16	4	256	128
2×10^{19}	12	640	16	4	256	160
6×10^{19}	15	768	16	4	256	192

Shared-expert-ratio candidates							
Candidate	1	2	3	4	5	6	7
n_a	12	11	10	8	6	4	2
n_s	0	1	2	4	6	8	10
S (%)	0	8.3	16.7	33.3	50	66.7	83.3

B.5 Case Studies

We present side-by-side comparisons of generations from LLaDA MoE v2 and SDAR Chat on identical prompts, taken from the mathematical and code benchmarks of Table 4. Both models generate under the decoding configurations of Appendix B.4. The comparisons are shown in Tables 14–17.

Table 14: **Case study on OlympiadBench (1).**

Case 1 OlympiadBench

Prompt.

You are a helpful assistant.
The quadratic polynomial $f(x)$ has a zero at $x = 2$. The polynomial $f(f(x))$ has only one real zero, at $x = 5$. Compute $f(0)$.
Please reason step by step, and put your final answer within \boxed{}.

Reference answer. $-\frac{32}{9}$

LLaDA MoE v2

We are given:
• $f(x)$ is a quadratic polynomial.
• $f(x)$ has a zero at $x = 2$.
• The composition $f(f(x))$ has only one real zero, at $x = 5$.
• We are to compute $f(0)$.

Step 1: General form of $f(x)$.
Since $f(x)$ is quadratic and has a zero at $x = 2$, we can write

$$f(x) = a(x - 2)(x - r)$$

for some constants $a \neq 0$ and r (the other root).

Step 2: Analyze $f(f(x))$.
We are told that $f(f(x))$ has only one real zero, at $x = 5$. That means $f(f(x)) = 0$ has only one real solution: $x = 5$. But $f(f(x)) = 0$ means $f(x)$ is a root of f, i.e., $f(x) = 2$ or $f(x) = r$. So

$$f(f(x)) = 0 \iff f(x) = 2 \quad \text{or} \quad f(x) = r$$

Thus, the zeros of $f(f(x))$ are the solutions to $f(x) = 2$ or $f(x) = r$. Among all these solutions, only $x = 5$ is real. Let's analyze this.

Step 3: Use the fact that $x = 5$ is a zero of $f(f(x))$.
So $f(f(5)) = 0$. That means $f(5)$ must be a root of f, i.e., $f(5) = 2$ or $f(5) = r$. But since $x = 5$ is the only real zero of $f(f(x))$, there must be no other real solutions to $f(x) = 2$ or $f(x) = r$. So let's consider possibilities.

Step 4: Suppose $f(5) = 2$.
Then $f(f(5)) = f(2) = 0$, so $x = 5$ is a zero of $f(f(x))$, as required. Now, we must ensure that no other real x satisfies $f(x) = 2$ or $f(x) = r$. But $f(x) = 2$ is a quadratic equation, so it can have at most 2 real solutions. Similarly, $f(x) = r$ can have at most 2 real solutions. We want only one real solution to $f(f(x)) = 0$, which is $x = 5$. So:
• $f(x) = 2$ must have only one real solution, which is $x = 5$;
• $f(x) = r$ must have no real solutions.

Step 5: Analyze $f(x) = 2$.
We want $f(x) = 2$ to have exactly one real solution, which is $x = 5$. But $f(x) = 2$ is a quadratic equation: $a(x - 2)(x - r) - 2 = 0$. This is a quadratic in x. For it to have exactly one real solution, **the discriminant must be zero, i.e., $f(x) = 2$ must have a double root at $x = 5$.** That means both $f(5) = 2$ and $f'(5) = 0$, because a double root implies the derivative is zero there. Let's use this.

Step 6: Use derivative condition.
We have $f(x) = a(x - 2)(x - r)$, so

$$f'(x) = a[(x - r) + (x - 2)] = a(2x - r - 2)$$

Setting $f'(5) = 0$ gives $a(2 \cdot 5 - r - 2) = a(8 - r) = 0$. Since $a \neq 0$, we get $r = 8$, so the other root of $f(x)$ is $x = 8$, and $f(x) = a(x - 2)(x - 8)$. Now use $f(5) = 2$: $f(5) = a(3)(-3) = -9a$. Setting $-9a = 2$ gives $a = -\frac{2}{9}$, so

$$f(x) = -\frac{2}{9}(x - 2)(x - 8)$$

Now compute $f(0)$:

$$f(0) = -\frac{2}{9}(-2)(-8) = -\frac{32}{9}$$

Step 7: Verify that $f(f(x))$ has only one real zero at $x = 5$.
We need to confirm that $f(f(x)) = 0$ has only one real solution, at $x = 5$. Recall $f(f(x)) = 0 \iff f(x) = 2$ or $f(x) = 8$. We already ensured that $f(x) = 2$ has a double root at $x = 5$, so it has only one real solution. Now check $f(x) = 8$: $-\frac{2}{9}(x - 2)(x - 8) = 8$ gives $(x - 2)(x - 8) = -36$, i.e., $x^2 - 10x + 52 = 0$, whose discriminant is $(-10)^2 - 4 \cdot 52 = -108 < 0$. So there are no real solutions to $f(x) = 8$. Therefore, the only real solution to $f(f(x)) = 0$ is $x = 5$, as required.

Final answer.

$$f(0) = -\frac{2}{9} \cdot 16 = \boxed{-\frac{32}{9}}$$

Correct

SDAR Chat

Given that $f(x)$ is a quadratic polynomial with a zero at $x = 2$, we can express $f(x)$ in the form:

$$f(x) = a(x - 2)^2 + b(x - 2) + c$$

where a, b, and c are constants.
We are also given that $f(f(x))$ has only one real zero at $x = 5$. This implies that $f(5) = 2$, since $f(x)$ must map $x = 5$ to a zero of $f(x)$.
First, we substitute $f(5) = 2$ into the polynomial:

$$f(5) = a(5 - 2)^2 + b(5 - 2) + c = 2$$

Simplifying, we get:

$$9a + 3b + c = 2 \quad (1)$$

Next, we need to find $f(0)$. We substitute $x = 0$ into the polynomial:

$$f(0) = a(0 - 2)^2 + b(0 - 2) + c = 4a - 2b + c$$

To find $f(0)$, we need to determine the values of a, b, and c. We already have one equation from $f(5) = 2$. We need another equation to solve for these variables. Since $f(f(x))$ has only one real zero at $x = 5$, we know that $f(x)$ must be such that $f(x) = 2$ has only one solution. **This implies that $f(x)$ is a perfect square trinomial.**
Assuming $f(x) = a(x - 2)^2$ (since it is a perfect square trinomial), we have:

$$f(x) = a(x - 2)^2$$

Substituting $x = 5$: $f(5) = a(5 - 2)^2 = 9a$. Given $f(5) = 2$: $9a = 2$, so $a = \frac{2}{9}$.
Thus, the polynomial $f(x)$ is:

$$f(x) = \frac{2}{9}(x - 2)^2$$

Now, we compute $f(0)$:

$$f(0) = \frac{2}{9}(0 - 2)^2 = \frac{2}{9} \cdot 4 = \frac{8}{9}$$

Therefore, the value of $f(0)$ is:

$$\boxed{\frac{8}{9}}$$

Incorrect

Table 15: **Case study on OlympiadBench (2).**

Case 2 OlympiadBench

Prompt.

You are a helpful assistant.

Let $T = 17$. In ARMLovia, the unit of currency is the edwah. Janet's wallet contains bills in denominations of 20 and 80 edwahs. If the bills are worth an average of $2T$ edwahs each, compute the smallest possible value of the bills in Janet's wallet.

Please reason step by step, and put your final answer within \boxed{}.

Reference answer. 1020

LLaDA MoE v2

We are given:

• $T = 17$.

• Janet's wallet contains bills in denominations of 20 and 80 edwahs.

• The average value of the bills is $2T = 2 \times 17 = 34$ edwahs.

• We are to find the smallest possible total value of the bills in the wallet.

Step 1: Define variables.

Let x be the number of 20-edwah bills and y the number of 80-edwah bills. Then the total number of bills is $x + y$, the total value is $20x + 80y$, and the average value is

$$\frac{20x + 80y}{x + y} = 34$$

Multiplying both sides by $x + y$ and expanding the right-hand side:

$$20x + 80y = 34x + 34y$$

Bringing all terms to one side gives $-14x + 46y = 0$, so $46y = 14x$, i.e.,

$$23y = 7x \quad \Longrightarrow \quad \frac{x}{y} = \frac{23}{7}$$

Since x and y must be integers, x must be a multiple of 23 and y a multiple of 7: $x = 23k$, $y = 7k$ for some positive integer k.

Step 2: Compute the total value.

$$20x + 80y = 20(23k) + 80(7k)$$

$$= 460k + 560k = 1020k$$

We want the smallest possible total value, so take $k = 1$: the total value is 1020 edwahs.

Step 3: Verify.

Number of bills: $x + y = 23 + 7 = 30$; total value: $20 \times 23 + 80 \times 7 = 460 + 560 = 1020$; average: $\frac{1020}{30} = 34$, as required.

Final answer.

$$\boxed{1020}$$

Correct

SDAR Chat

Given that $T = 17$, the average value of the bills in Janet's wallet is $2T = 2 \times 17 = 34$ edwahs each.

Let x be the number of 20-edwah bills and y the number of 80-edwah bills. The total value of the bills is $20x + 80y$, the total number of bills is $x + y$, and the average value is

$$\frac{20x + 80y}{x + y} = 34$$

Setting up the equation and simplifying:

$$20x + 80y = 34x + 34y$$

$$80y - 34y = 34x - 20x$$

so $46y = 14x$. Dividing both sides by 2:

$$23y = 7x$$

To find the smallest possible value of the bills, we need the smallest positive integers x and y that satisfy this equation. **The smallest solution is $y = 23$ and $x = 7$.**

Substituting these values back into the total value equation:

$$20x + 80y = 20(7) + 80(23) = 140 + 1840 = 1980$$

Thus, the smallest possible value of the bills in Janet's wallet is:

$$\boxed{1980}$$

Incorrect

Table 16: **Case study on MultiPL-E (1).**

Case 3 MultiPL-E

Prompt.

```
#include <assert.h>
#include <bits/stdc++.h>
// From a given vector of integers, generate a vector of rolling maximum
// element found until given moment in the sequence.
// >>> rolling_max((std::vector<long>({(long)1, (long)2, (long)3, (long)2,
//    (long)3, (long)4, (long)2})))
// (std::vector<long>({(long)1, (long)2, (long)3, (long)3, (long)3, (long)4, (long)4}))
std::vector<long> rolling_max(std::vector<long> numbers) {
```

Reference tests. The benchmark test cases for this problem.

LLaDA MoE v2

```
if (numbers.empty()) {
  return {};
}
std::vector<long> result;
long current_max = numbers[0];
result.push_back(current_max);
for (size_t i = 1; i < numbers.size(); ++i) {
  if (numbers[i] > current_max) {
    current_max = numbers[i];
  }
  result.push_back(current_max);
}
return result;
```

Pass

SDAR Chat

```
std::vector<long> result;
long max_num = numbers[0];
  // no empty-input check:  numbers[0]
  // is out of bounds on an empty vector
result.push_back(max_num);
for (int i = 1; i < numbers.size(); i++) {
  if (numbers[i] > max_num) {
    max_num = numbers[i];
  }
  result.push_back(max_num);
}
return result;
```

Fail

Table 17: **Case study on MultiPL-E (2).**

Case 4 MultiPL-E

Prompt.

```
//Return 2^n modulo p (be aware of numerics).
// >>> modp(3, 5)
// 3
// >>> modp(1101, 101)
// 2
// >>> modp(0, 101)
// 1
// >>> modp(3, 11)
// 8
// >>> modp(100, 101)
// 1
function modp(n: number, p: number): number {
```

Reference tests. The benchmark test cases for this problem.

LLaDA MoE v2

```
// Handle edge cases
if (p === 1) return 0;
if (n === 0) return 1;
// Use modular exponentiation
// (binary exponentiation)
let result = 1;
let base = 2 % p; // Start with 2 modulo p
while (n > 0) {
  // If n is odd, multiply result
  // by current base
  if (n % 2 === 1) {
    result = (result * base) % p;
  }
  // Square the base and halve the exponent
  base = (base * base) % p;
  n = Math.floor(n / 2);
}
return result;
```

Pass

SDAR Chat

```
// Initialize result to 1
let res = 1;
// Update n if it is more than or equal to p
n = n % p;
  // wrong:  2^n mod p ≠ 2^(n mod p) mod p
// If n is 0, then any power of n is 0
if (n === 0) {
  return 0;
}
// wrong:  2^0 = 1, so modp(0, p) = 1
// Main loop to calculate 2^n modulo p
while (n > 0) {
  // If n is odd, multiply res by 2
  // and take modulo p
  if (n & 1) {
    res = (res * 2) % p;
  }
  // no base squaring:  this gives
  // 2^popcount(n) mod p, not 2^n mod p
  // Divide n by 2
  n = n >> 1;
}
return res;
```

Fail