ERMINE: NEURAL DIFFERENTIAL SCALING LAWS FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling laws prescribe how to spend compute; for diffusion language models (DLMs), they should also prescribe *when*. Existing compute-optimal recipes for DLMs inherit the Chinchilla scaling law, which allocates model capacity and data uniformly across the diffusion process. We observe, however, that DLM loss varies substantially with the diffusion timestep, leaving compute underutilized under Chinchilla-style allocation. We introduce *Ermine*, a differential scaling law that characterizes the instantaneous loss via time-dependent model capacity and data densities. We derive its optimal solution under a fixed compute budget and realize it with quantile-based sampling and a sparse Mixture-of-Experts (MoE) architecture. Ermine accelerates Chinchilla-optimal training by $1.35\times$, improves compute efficiency over dense DLMs by $1.87\times$, and narrows the compute-optimal gap between DLMs and autoregressive models (ARMs) from $17.56\times$ to $9.57\times$. On OpenWebText, our model attains a state-of-the-art validation perplexity of 17.63 among DLMs, and at 1.1×10^{21} FLOPs it outperforms existing ARMs and DLMs on five zero-shot commonsense reasoning benchmarks using only 30-50% of their training FLOPs. Our analysis further reveals that Ermine generalizes better by avoiding excessive compute in time regions where scalability diminishes.

1 INTRODUCTION

The compute required to train large language models (LLMs) has become one of the fastest-growing expenditures in the technology industry. As characterized by neural scaling laws (Hestness et al., 2017; Rosenfeld et al., 2020; Kaplan et al., 2020; Hoffmann et al., 2022), improving LLM capabilities demands exponential increases in computing power. The Chinchilla law (Hoffmann et al., 2022) identifies *model capacity* (N) and *data volume* (D) as the dominant factors governing the final loss, and determines their optimal allocation under a fixed compute budget (C).

Diffusion language models (DLMs) have recently emerged as a scalable alternative to autoregressive models (ARMs) for building LLMs, and prior works (Ni et al., 2025; Chao et al., 2026; Rütte et al., 2026) have applied the Chinchilla law to derive compute-optimal training recipes for them. However, the two model families differ in how they decompose generation: ARMs factorize the complexity of sequence generation over token positions, while DLMs distribute it along a time axis t. This difference manifests empirically at scale. As shown in Fig. 2 (c), the loss of Chinchilla-optimal ARMs improves nearly uniformly across positions as compute grows, while that of DLMs (including masked diffusion models (MDMs) and uniform-state diffusion models (UDMs)) varies substantially with the timestep, revealing different degrees of scalability *within* the diffusion process. This non-uniformity implies that

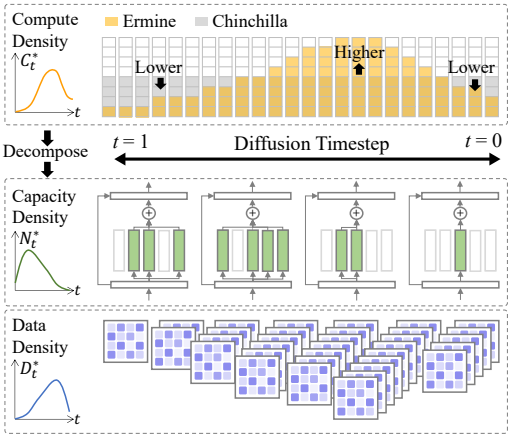

Figure 1: Illustration of Ermine, which reallocates compute across diffusion timesteps (top; vs. Chinchilla's uniform allocation) and decomposes the compute density C_t^* into capacity density N_t^* (middle) and data density D_t^* (bottom).

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

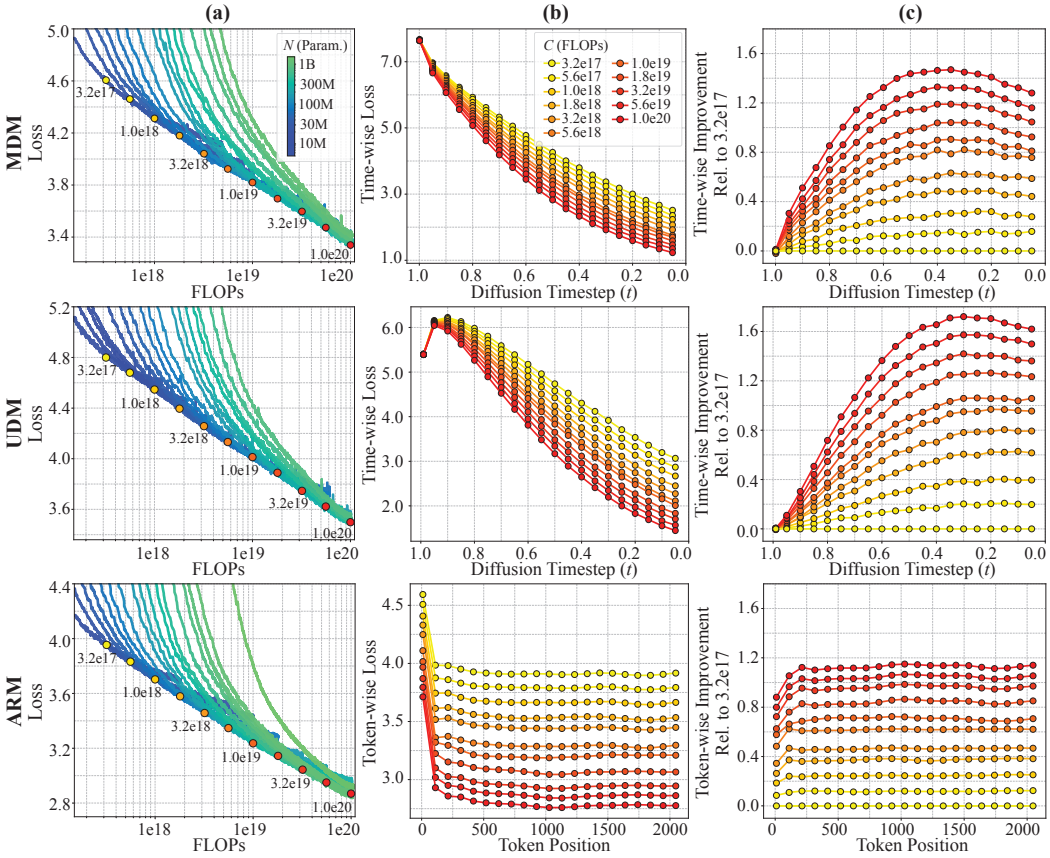

Figure 2: **(a)** Training loss versus FLOPs, with compute-optimal configurations marked red and yellow along the frontier. Model size (N) varies from 5M to 1.5B parameters. **(b)** Time-wise losses for DLM (MDM and UDM) and token-wise losses for ARM under compute-optimal setups from $C = 3.2 \times 10^{17}$ to 1.0×10^{20} FLOPs. **(c)** Time-wise and token-wise loss improvement relative to the smallest budget ($C = 3.2 \times 10^{17}$ FLOPs). Improvement for MDM and UDM concentrates in the intermediate-time regime, whereas that of ARM remains nearly uniform across token positions.

Chinchilla-style allocation leaves compute underutilized, motivating a redefinition of the scaling law that redistributes compute toward high-improvement time regions.

The time-wise improvements of DLMs (Fig. 2 (c)) follow a consistent, predictable pattern in t and C. To capture this, we introduce *Ermine*[1], a differential scaling law that models the instantaneous loss $\hat{\ell}_t(N_t, D_t)$ via time-dependent capacity density N_t and data density D_t. Solving Ermine under a fixed budget C yields a time-dependent training recipe: the optimal compute density C_t^* and its corresponding decomposition into D_t^* and N_t^* (Fig. 1). We realize this design with a quantile-based sampling to concentrate data in targeted time regimes, and a sparse Mixture-of-Experts (MoE) (Shazeer et al., 2017) architecture to vary active capacity over time. Experiments show that Ermine improves training efficiency, pushes the compute-optimal Pareto frontier outward (Fig. 7), and strengthens generalization and zero-shot performance. Our contributions are as follows:

- We introduce Ermine, the first scaling law to characterize the time-wise scaling behavior of DLMs and prescribe the optimal allocation of compute, data, and capacity.
- Ermine accelerates Chinchilla-optimal training by $1.35\times$ and yields a model $1.87\times$ more compute-efficient than a dense DLM, narrowing the compute-optimal gap between DLM and ARM from $17.56\times$ to $9.57\times$. Under matched training FLOPs, it achieves state-of-the-art perplexity on OpenWebText (17.63), approaching the ARM result (17.54) on benchmark.
- At 1.1×10^{21} FLOPs, our model outperforms GPT-Neo (Black et al., 2022), OPT (Zhang et al., 2022), Pythia (Biderman et al., 2023), BLOOM (BigScience, 2023), TinyLlama (Zhang et al., 2024), SMDM (Nie et al., 2025), and MDM-Prime-v2 (Chao et al., 2026) on five zero-shot commonsense reasoning benchmarks using only 30–50% of their training FLOPs.

[1]The ermine changes its coat with the seasons, as our law adapts capacity and data over diffusion time.

2 Background and Related Work

This section provides background on Chinchilla scaling laws (Section 2.1) and DLMs (Section 2.2), and closes with a brief discussion of how Ermine differs from prior work (Section 2.3).

2.1 Chinchilla Scaling Law

Let N denote the model parameter count, D the number of training tokens, (E, A, B, a, b) a set of learnable coefficients. The Chinchilla scaling law (Hoffmann et al., 2022) determines how a fixed compute budget $C \approx 6ND$ (measured in floating-point operations (FLOPs)) should be optimally allocated between N and D to minimize the loss $\hat{\mathcal{L}}$, which is characterized by the following form:

$$\hat{\mathcal{L}}(N, D) \triangleq E + AN^{-a} + BD^{-b}, \tag{1}$$

where E represents the irreducible loss, and the terms AN^{-a} and BD^{-b} capture the reducible penalties from finite model capacity and finite data, respectively. In the maximum likelihood paradigm, E and $AN^{-a} + BD^{-b}$ correspond to the data entropy and the KL divergence between the data and parameterized distributions, respectively. When visualized in log space, the exponent (a or b) of each penalty term governs the *slope* at which the penalty diminishes with scale, while the coefficient (A or B) sets its constant *shift*, i.e., $\log(AN^{-a}) = \log A - a \log N$, where $\log A$ is the vertical shift and a is the slope.

The coefficients (E, A, B, a, b) are estimated from a suite of M training runs $\{(N^i, D^i, \mathcal{L}^i)\}_{i=1}^{M}$ spanning a range of compute scales, by minimizing a Huber-loss objective (Huber, 1964) (see Eq. (A11) in Appendix A.4.1). Given the fitted coefficients and a target budget C, the compute-optimal allocation follows in closed form from the constrained problem:

$$\min_{N,\, D} \hat{\mathcal{L}}(N, D) \quad \text{s.t.} \quad 6ND = C. \tag{2}$$

The solutions to Eq. (2) are expressed as functions of C:

$$N^* = G \left(\frac{C}{6}\right)^{\frac{b}{a+b}} \propto C^{\frac{b}{a+b}}, \qquad D^* = G^{-1} \left(\frac{C}{6}\right)^{\frac{a}{a+b}} \propto C^{\frac{a}{a+b}}, \tag{3}$$

where $G \triangleq \left(\frac{aA}{bB}\right)^{\frac{1}{a+b}}$ represents the *split ratio* (i.e., $\frac{D^*}{N^*} = G^{-2}(C/6)^{\frac{a-b}{a+b}} \approx G^{-2}$ when $a \approx b$) and determines the optimal ratio of training tokens to parameters. The capacity- and data-split exponents $\frac{b}{a+b}$ and $\frac{a}{a+b}$ are fractional numbers dictating how additional compute is distributed between a larger model and more tokens. By substituting (N^*, D^*) back into Eq. (2), the two reducible terms further collapse onto a single combined power of C, $\hat{\mathcal{L}}(C) \triangleq \hat{\mathcal{L}}(N^*, D^*) = E + FC^{-\gamma}$, with $\gamma \triangleq \frac{ab}{a+b}$ and $F \triangleq 6^{\gamma}[AG^{-a} + BG^{b}]$ (Kaplan et al., 2020; Pearce & Song, 2024) representing, respectively, the slope of the compute-loss frontier in log space and its constant shift.

2.2 Diffusion Language Models

Let $\boldsymbol{x} = [x^1, \cdots, x^L] \in \mathcal{X}^L$ be a sequence of L tokens, where x^i denotes an element of $\boldsymbol{x}$ and $\mathcal{X} \triangleq \{1, \cdots, V\}$ represents a set of V token indices. Given a time variable $t \in [0, 1]$, $\boldsymbol{x}_0 \in \mathcal{X}^L$ denotes a sample drawn from the data distribution q, and $\boldsymbol{x}_t \in \mathcal{X}^L$ denotes the latent variable introduced by the forward diffusion process. Let $\delta_{x'}(x)$ be the Kronecker delta function, which equals 1 if $x = x'$ and 0 otherwise.

Forward and Reverse Processes. The forward process is a Markov chain in t that perturbs each token independently, $q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \prod_{i=1}^{L} q(x_t^i|x_0^i)$, determined by a strictly decreasing schedule $\alpha : [0, 1] \to [0, 1]$, whose value α_t is the probability that a token remains unperturbed. Its element-wise kernel interpolates between $\delta_{x_0^i}(x_t^i)$ and a fixed prior π on $\mathcal{X}$ (Sahoo et al., 2024; Shi et al., 2024):

$$q(x_t^i|x_0^i) = \alpha_t \delta_{x_0^i}(x_t^i) + (1 - \alpha_t)\pi(x_t^i). \tag{4}$$

Since $\alpha_0 = 1$ and $\alpha_1 = 0$, the kernel in Eq. (4) reduces to $\delta_{x_0^i}$ at the *clean end* ($t = 0$) and to π at the *noisy end* ($t = 1$), where $\boldsymbol{x}_1$ is independent of $\boldsymbol{x}_0$. Writing $\alpha_{t|s} \triangleq \alpha_t/\alpha_s$ for $s < t$, the transition kernel is $q(x_t^i|x_s^i) = \alpha_{t|s}\delta_{x_s^i}(x_t^i) + (1 - \alpha_{t|s})\pi(x_t^i)$. Bayes' rule yields the posterior

$q(x_s^i|x_t^i, x_0^i) = q(x_t^i|x_s^i)\, q(x_s^i|x_0^i)/q(x_t^i|x_0^i)$. The reverse-time model $p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)$ is defined through *mean parameterization* (Shi et al., 2024) by replacing $\delta_{x_0^i}$ in $q(x_s^i|x_t^i, x_0^i)$ with a softmax distribution of x_0^i predicted from the noisy sequence $\boldsymbol{x}_t$ by a network parameterized by θ.

Objective. The objective is the negative evidence lower bound (ELBO) on the negative log likelihood $\mathbb{E}_{q(\boldsymbol{x})}[-\log p_\theta(\boldsymbol{x})]$, which decomposes into reconstruction, diffusion, and prior terms (Austin et al., 2021) over a discretization of step size $1/T$. Under the mean parameterization with the terminal conditions at $t = 0$ and 1, the loss can be simplified and only the diffusion term remains (Sahoo et al., 2024; Schiff et al., 2025). With $s \triangleq t - 1/T$, the loss reduces to a single time integral:

$$\mathcal{L} = \int_0^1 \lim_{T\to\infty} T \cdot \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[D_{\mathrm{KL}}\left[q(\boldsymbol{x}_s|\boldsymbol{x}_t, \boldsymbol{x}_0) \,\middle\|\, p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\right]\right] dt \triangleq \int_0^1 \ell_t\, dt, \tag{5}$$

where $q(\boldsymbol{x}_0, \boldsymbol{x}_t) \triangleq q(\boldsymbol{x}_0)\, q(\boldsymbol{x}_t|\boldsymbol{x}_0)$, and ℓ_t denotes the time-wise loss. The choice of π is the design decision that separates the DLM families, and it determines the closed form of ℓ_t. Setting $\pi = \delta_{\mathrm{m}}$ for a special mask token m recovers MDMs (Eq. (A7)), while $\pi = 1/V$ recovers UDMs (Eq. (A8)). The closed-form formulations of ℓ_t^{MDM} and ℓ_t^{UDM} are summarized in Appendix A.2.

2.3 Related Work

We discuss related work along two groups. One covers methods that allocate non-uniform compute across diffusion processes, and the other includes scaling studies of diffusion models. Tables A3 and A4 in Appendix A.3 summarize how Ermine differs from each group, respectively.

The first group (Karras et al., 2022; Kingma & Gao, 2023; Hang et al., 2024; Hong et al., 2026; Zhang et al., 2026) determines the time-dependent data sampling or model size through a hand-designed schedule (e.g., Gaussian, Laplace). This line of studies has two limitations. First, the functional form is fixed in advance, so its hyperparameters (e.g., the mean and variance of a Gaussian) must be obtained via grid search. Second, none of these methods analyzes the scaling behavior in N and D, and the resulting schedule is not adaptive to the budget C. The second group (Nie et al., 2025; Ni et al., 2025; Liang et al., 2026; Chickering et al., 2026) treats compute as a time-independent scalar, assuming every diffusion timestep is equally worth scaling. Ermine addresses both groups' limitations. Unlike the first group, Ermine fits its allocation using scaling laws rather than hand-designing it, making it free of hyperparameters and scalable with compute (C-dependent). Unlike the second group, Ermine captures the time-dependency of scaling laws. Section 4.1 and Appendix A.6.1 (**AS4-5**) show that Ermine outperforms both Chinchilla and hand-designed allocations.

3 Methodology

This section introduces *Ermine*, a time-dependent generalization of Chinchilla. Section 3.1 elaborates on the parameterization and its optimal solution. Section 3.2 presents the practical realization.

3.1 Ermine: Neural Differential Scaling Laws

To characterize the time-wise loss scaling behavior of DLMs, we generalize Eq. (1) to a time-dependent setting, where the loss integrand varies along $t \in [0, 1]$. Ermine is parameterized as:

$$\hat{\mathcal{L}}(N_t, D_t) \triangleq \int_0^1 \hat{\ell}_t(N_t, D_t)\, dt = \int_0^1 \left(E_t + A_t N_t^{-a_t} + B_t D_t^{-b_t}\right) dt, \tag{6}$$

where N_t and D_t denote the parameter and data *densities* at time t. In Eq. (6), E_t and $A_t N_t^{-a_t} + B_t D_t^{-b_t}$ capture the time-wise irreducible and reducible losses, respectively, which correspond to the following KL divergence terms based on the decomposition of the time-wise loss in Eq. (5):

$$\begin{aligned}&\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\left[q(\boldsymbol{x}_s|\boldsymbol{x}_t, \boldsymbol{x}_0) \,\|\, p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\right]\Big] \\ &= \underbrace{\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\left[q(\boldsymbol{x}_s|\boldsymbol{x}_t, \boldsymbol{x}_0) \,\|\, q(\boldsymbol{x}_s|\boldsymbol{x}_t)\right]\Big]}_{\text{irreducible}} + \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\left[q(\boldsymbol{x}_s|\boldsymbol{x}_t) \,\|\, p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\right]\Big]}_{\text{reducible}}.\end{aligned} \tag{7}$$

The derivation is in **Proposition** A2. Its difference from Chinchilla is summarized in Table 1.

Table 1: Comparison between the Chinchilla and Ermine scaling laws.

	Chinchilla (Eq. (1))	Ermine (Eq. (6))
Irreducible Term	$\mathbb{E}_{q(\boldsymbol{x})}\left[-\log q(\boldsymbol{x})\right]$	$\lim_{T\to\infty} T \cdot \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[D_{\mathrm{KL}}\left[q(\boldsymbol{x}_s\|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,q(\boldsymbol{x}_s\|\boldsymbol{x}_t)\right]\right]$
Parameterization	E	E_t
Reducible Term	$D_{\mathrm{KL}}\left[q(\boldsymbol{x})\,\|\,p_\theta(\boldsymbol{x})\right]$	$\lim_{T\to\infty} T \cdot \mathbb{E}_{q(\boldsymbol{x}_t)}\left[D_{\mathrm{KL}}\left[q(\boldsymbol{x}_s\|\boldsymbol{x}_t)\,\|\,p_\theta(\boldsymbol{x}_s\|\boldsymbol{x}_t)\right]\right]$
Parameterization	$AN^{-a}+BD^{-b}$	$A_tN_t^{-a_t}+B_tD_t^{-b_t}$

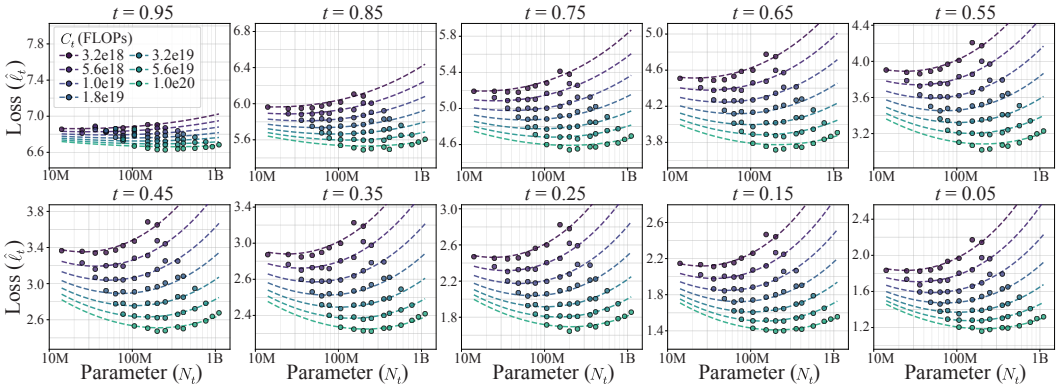

Figure 3: Time-wise scaling curves fitted by Ermine, obtained from MDM (MoE model) trained on C4 (Raffel et al., 2020). Each panel plots the time-wise loss $\hat{\ell}_t$ against N_t under a series of C_t, with the fitted parametric curves shown as dashed lines. The results of UDM are shown in Fig. A10.

Following the Chinchilla parameter-fitting procedure, the Ermine coefficients $(E_t, A_t, B_t, a_t, b_t)$ are estimated from a suite of M training runs $\{(N_t^i, D_t^i, \ell_t^i)\}_{i=1}^M, \forall t \in [0,1]$ by minimizing a Huber-loss objective (Eq. (A11)). Since the loss at each timestep is estimated with a single-step Monte Carlo (MC) sample, the compute density retains the standard form $C_t = 6N_tD_t$ (see Appendix A.4.2). The fitted coefficients (Fig. A11) yield scaling curves that vary with t (example shown in Fig. 3).

The total compute C consumed by the training is calculated as the integration of the compute density $C = \int_0^1 C_t\,dt = \int_0^1 6N_tD_t\,dt$ (see Appendix A.4.2). To identify the optimal allocation of N_t and D_t given a target (global) budget C, the optimization seeks the solution of the following formula:

$$\min_{N_t,D_t} \hat{\mathcal{L}}(N_t, D_t) \quad \text{s.t.} \quad \int_0^1 6N_tD_t\,dt = C. \tag{8}$$

Unlike Eq. (2), Eq. (8) specifies an integral in the constraint and (N_t^*, D_t^*) is derived through the introduction of a Lagrange multiplier μ. Ermine considers the following Lagrangian:

$$\mathcal{J}(N_t, D_t, \mu) = \int_0^1 \left(E_t + A_tN_t^{-a_t} + B_tD_t^{-b_t}\right) dt + \mu\left(\int_0^1 6N_tD_t dt - C\right). \tag{9}$$

Let $G_t \triangleq \left(\frac{a_tA_t}{b_tB_t}\right)^{\frac{1}{a_t+b_t}}$, $\gamma_t \triangleq \frac{a_tb_t}{a_t+b_t}$, and $F_t \triangleq 6^{\gamma_t}(A_tG_t^{-a_t} + B_tG_t^{b_t})$, defined in a similar way as Chinchilla (Section 2.1). The optimal solution to Eq. (9) is given in **Proposition** 1.

Proposition 1. *Given a target compute budget C, the optimal solution that minimizes Eq. (9) is:*

$$C_t^* = \left(\frac{\gamma_tF_t}{\mu^*}\right)^{\frac{1}{1+\gamma_t}}, \quad N_t^* = G_t\left(\frac{C_t^*}{6}\right)^{\frac{b_t}{a_t+b_t}}, \quad D_t^* = G_t^{-1}\left(\frac{C_t^*}{6}\right)^{\frac{a_t}{a_t+b_t}}, \tag{10}$$

where μ^* is unique and satisfies $\int_0^1 (\gamma_tF_t/\mu)^{\frac{1}{1+\gamma_t}}\,dt = C$.

The proof is in Appendix A.1.1. The proposition guarantees that the solution is unique. We obtain μ^* with a bisection solver (Kaw et al., 2009) (Algorithm 1). Given μ^*, the compute density C_t^* is determined by Eq. (10), and splits into $N_t^* \propto (C_t^*)^{\frac{b_t}{a_t+b_t}}$ and $D_t^* \propto (C_t^*)^{\frac{a_t}{a_t+b_t}}$ as in Chinchilla (Eq. (3)). Fig. 4 **(1)-(4)** visualizes $(C_t^*, N_t^*, D_t^*, \mu^*)$ obtained from **Proposition** 1 across budgets C.

Interpretation of Optimal Compute Density. C_t^* is the term that differs from Chinchilla, and its form can be understood via an exchange argument. If an additional FLOP reduces the time-wise loss $\hat{\ell}_t$ more at one timestep than at another, moving a FLOP from the latter to the former would lower

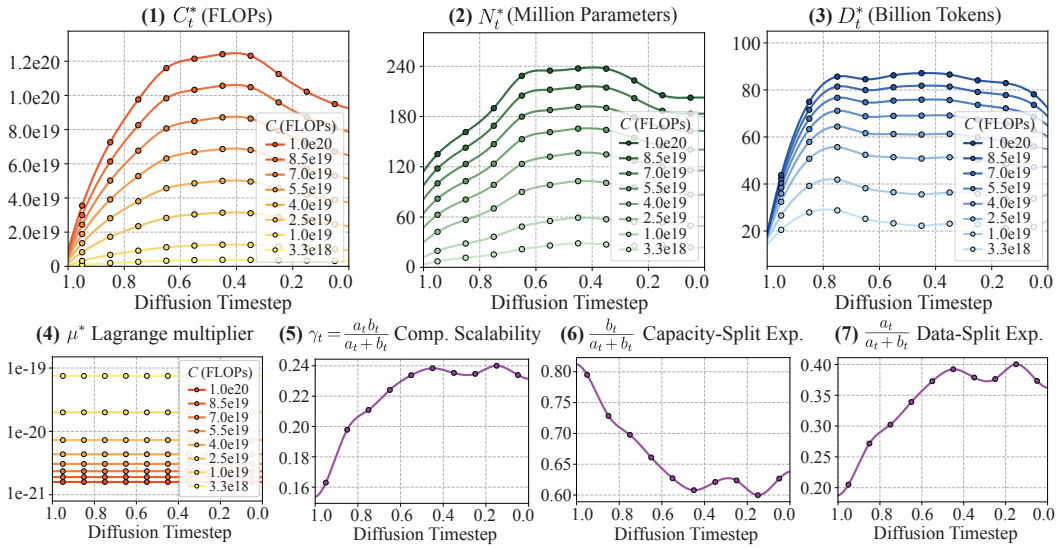

Figure 4: **(1)-(3)** Analytic solutions of C_t^*, N_t^*, and D_t^* derived from Eq. (10), with compute budget C ranging from 3.3×10^{18} to 1.0×10^{20} FLOPs. **(4)** μ^* is the solution to $\int_0^1 (\gamma_t F_t/\mu)^{\frac{1}{1+\gamma_t}}\, dt = C$ and shrinks as C grows. **(5)** Compute scalability γ_t, representing the exponent of C_t in the time-wise loss ($\hat{\ell}_t(C_t) = E_t + F_t C_t^{-\gamma_t}$). **(6)-(7)** The split exponents $b_t/(a_t + b_t)$ and $a_t/(a_t + b_t)$. This example uses MDM-MoE; other variants appear in Figs. A8, A12, and A13.

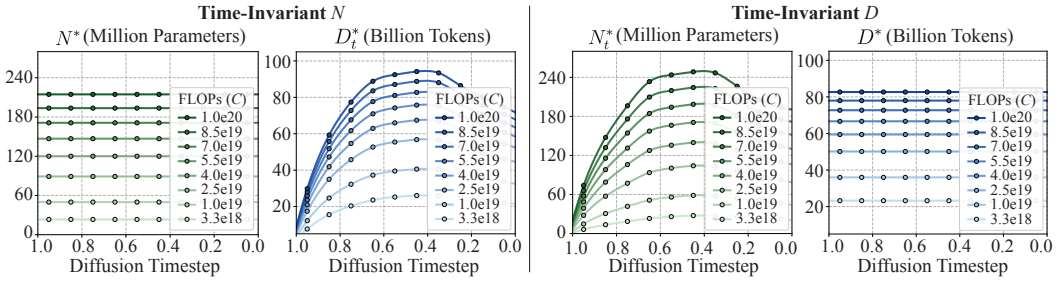

Figure 5: Solutions for Ermine in the time-invariant N and time-invariant D cases.

the total loss $\hat{\mathcal{L}}$ within the same budget. At the optimum, no such move is possible. An additional FLOP must reduce the loss by the same amount, denoted μ^*, at every timestep (i.e., the stationarity condition of Eq. (9)). For $\hat{\ell}_t(C_t) = E_t + F_t C_t^{-\gamma_t}$, this reduction is $-\, d\hat{\ell}_t/dC_t = \gamma_t F_t C_t^{-(1+\gamma_t)}$, and equating it to μ^* yields C_t^* in Eq. (10). Each factor of C_t^* has a role. The product $\gamma_t F_t$ sets the scale of $-\, d\hat{\ell}_t/dC_t$, meaning more productive timesteps receive more budget. The multiplier μ^* is the time-independent value that $\gamma_t F_t$ balances against. It scales the allocation to meet the budget $\int_0^1 C_t^*\, dt = C$, and a larger C lowers μ^* and raises C_t^* simultaneously for all t. Finally, the exponent $1/(1+\gamma_t) \in (0, 1]$ (empirically ≈ 1) mildly compresses differences in $\gamma_t F_t/\mu^*$ across timesteps.

Empirical Trends. Across MDMs and UDMs implemented with MoE and dense Transformer architectures, we observe three consistent trends **T1-3** (see Fig. A8 and Appendix A.6.4):

- **(T1) The Optimal Compute Density is Unimodal.** C_t^* peaks at $t \in [0.4, 0.5]$, indicating that samples mixing noisy and clean tokens in comparable proportions demand the most compute to denoise. The peak position changes according to α_t in the diffusion kernel (Eq. (4)), and can be located using the signal-to-noise ratio: $\log \frac{\alpha_t}{1-\alpha_t}$ (see Fig. A9 and Appendix A.6.4).
- **(T2) Compute Scalability Diminishes at the Noisy End.** γ_t decreases as $t \to 1$ (Fig. 4 **(5)**), so allocating compute there barely improves performance. Models given excessive compute in this region overfit, and uniform allocation leads to worse generalization (see Section 4.4).
- **(T3) Noisier Timesteps Favor Capacity over Data.** The capacity and data allocation exponents $\frac{b_t}{a_t+b_t}$ and $\frac{a_t}{a_t+b_t}$ vary substantially across t (Fig. 4 **(6)**, **(7)**). As $t \to 1$, the capacity exponent dominates, indicating that larger models scale better than longer training. Correspondingly, smaller models overfit severely at the noisy end (see Appendix A.6.3 (**S3**)).

Allocation in Time-Invariant Cases. Ermine requires both N_t and D_t to vary with t. Certain architectures (e.g., dense Transformers) admit no per-timestep control over the model size. To cover these settings, we consider semi-time-dependent setups where only one quantity varies, i.e., $N_t = N$ or $D_t = D$, $\forall t \in [0, 1]$. **Proposition** A1 in Appendix A.4.1 gives the analytic solutions for ***(i)*** time-invariant N and ***(ii)*** time-invariant D, visualized in Fig. 5. Section 4.2 evaluates these cases, which are slightly inferior to the fully time-dependent allocation (Eq. (10)), and Appendix A.6.1 **(AS4-5)** shows that cases ***(i)*** and ***(ii)*** outperform prior hand-designed allocation methods.

3.2 Practical Implementation

This section describes how to realize Ermine allocation (Eq. (10)) under data parallelism.

Time-dependent Data Allocation (D_t^*). DLM training (e.g., (Sahoo et al., 2024)) segments the time axis according to the available units of compute (UoC) per training update to approximate the integration. Let u denote the total UoC (e.g., the number of GPUs or gradient accumulation steps). One samples $t_1 \sim \mathcal{U}[0, 1/u]$, $\ldots$, $t_u \sim \mathcal{U}[(u-1)/u, 1]$ and averages the losses, such that data are consumed uniformly over time. To realize a non-uniform D_t^*, we instead partition the time axis at the $1/u$-quantiles of the cumulative distribution function (CDF) of D_t^* and assign each resulting interval to one UoC. Denote these interval boundaries as $\{\tau_1, \ldots, \tau_{u-1}\}$ (with $\tau_0 = 0$, $\tau_u = 1$). The time variable is sampled as $t_1 \sim \mathcal{U}[0, \tau_1]$, $\ldots$, $t_u \sim \mathcal{U}[\tau_{u-1}, 1]$. Fig. 6 illustrates this construction for $u = 8$: intervals are narrower where D_t^* is large, allocating proportionally more data to these regions, and wider where D_t^* is small.

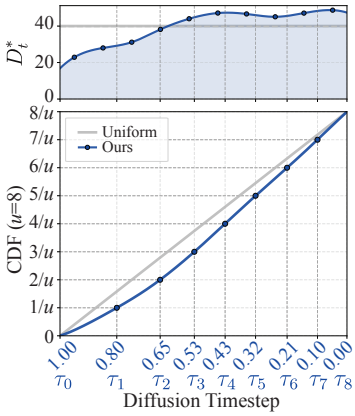

Figure 6: **Top**: D_t^* across diffusion timesteps. **Bottom**: CDF of D_t^* with $1/u$-quantile boundaries (blue) compared to uniform chunking (gray).

Time-dependent Parameter Allocation (N_t^*). To vary model capacity over time, we use an MoE architecture and set a per-interval top-k to match N_t^* (see Fig. A2). Using the same partition $\{\tau_1, \ldots, \tau_{u-1}\}$, we define the target averaged by interval $N_i^* = \frac{1}{\tau_i - \tau_{i-1}} \int_{\tau_{i-1}}^{\tau_i} N_t^* \, dt, \quad i \in \{1, \ldots, u\}$. The active-parameter count in MoE follows $N_i^* \triangleq N_0 + N_\Delta k_i$, where N_0 is the base parameter count and N_Δ is the per-expert increment. As a result, $k_i = \frac{N_i^* - N_0}{N_\Delta}$ for interval i. Appendix A.6.4 analyzes Ermine under dense and MoE architectures, and Appendix A.6.1 compares against a depth-wise realization (i.e., the Looped Transformer (Giannou et al., 2023) and its sparse variant), which MoE outperforms by a clear margin.

In summary, our training recipe reduces to an allocation configuration that takes (C, u) as input and returns $\{(\tau_{i-1}, \tau_i, k_i)\}_{i=1}^u$. Appendices A.4 and A.5 provide implementation details.

4 Experiments

We evaluate Ermine along four axes: the compute-loss frontier it attains relative to Chinchilla (Section 4.1), likelihood on the OpenWebText (OWT) benchmark (Section 4.2), zero-shot accuracy at larger scale (Section 4.3), and time-wise generalization (Section 4.4). Appendix A.5 provides training configurations, and Appendix A.6 presents 14 additional analyses (**AS1-5**, **TC1-3**, **S1-3**, **T1-3**) that further probe Ermine's effectiveness, wall-time efficiency, and generalizability. Appendix A.7 summarizes the limitations (**L1-3**) and future directions of this work.

4.1 Ermine Establishes a New Efficiency Frontier

We perform experiments on C4 (Raffel et al., 2020). We adopt LLaMA-styled (Touvron et al., 2023) architecture with eight total experts and two active experts in the MoE layer as the default setup. The total parameter count is $3\times$ the active parameter count (see Table A6). Following previous MoE scaling studies (Ludziejewski et al., 2025), FLOPs are calculated based on active parameters.

Fig. 7 depicts the efficiency frontiers of ARM (Chinchilla), UDM (Chinchilla), MDM (Chinchilla), MDM-MoE (Chinchilla), and MDM-MoE (Ermine), where each traces the validation loss of the

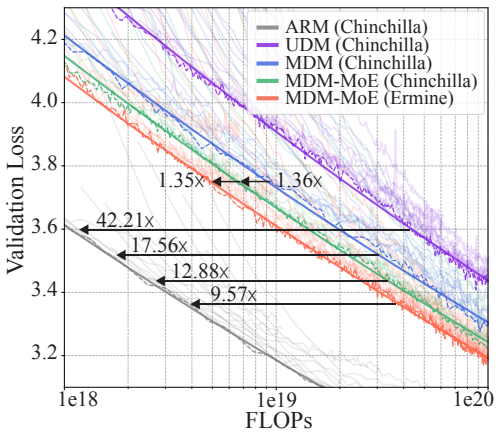

Figure 7: Efficiency frontiers on C4. Our model reduces the compute an MDM needs to match the ARM's loss from $17.56\times$ to $9.57\times$. MoE improves the compute efficiency of the dense MDM by $1.36\times$, and Ermine's allocation contributes a further $1.35\times$. Thin lines show individual runs, dashed lines their lower envelope, and solid lines the fitted power laws.

Table 2: PPL on OWT. Methods marked $*$ use the compute-optimal setup, with non-embedding (activated) parameters (N) and tokens (D) set by the frontier in Fig. A3. Total compute (C) is 2.89×10^{20} FLOPs. †Ermine's N and D are integrated values. Table A11 shows zero-shot PPL on seven unseen datasets.

	N	D	PPL (↓)
Order-Specific Models			
BD3-LMs (Arriola et al., 2025)	92M	524B	≤20.73
ARM (Sahoo et al., 2024)	92M	524B	17.54
ARM* (Chinchilla)	1251M	39B	**12.98**
Order-Agnostic Models			
UDM (Sahoo et al., 2025)	92M	524B	≤25.20
SEDD (Lou et al., 2024)	92M	524B	≤24.10
GenMD4 (Shi et al., 2024)	92M	524B	≤21.80
MDM (Sahoo et al., 2024)	92M	524B	≤22.98
MDM-MoE	92M	524B	≤21.44
MDM* (Chinchilla)	546M	89B	≤18.85
MDM-MoE* (Chinchilla)	466M	104B	≤18.15
MDM-MoE* (Ermine, Inv. D)	442M†	109B	≤17.72
MDM-MoE* (Ermine, Inv. N)	446M	108B†	≤17.70
MDM-MoE* (Ermine)	459M†	105B†	≤**17.63**

compute-optimal configuration. We highlight three observations. First, UDM is consistently inferior to MDM across all budgets, requiring $42.21\times$ more compute than ARM to reach the same validation loss, compared to $17.56\times$ for MDM. We therefore adopt MDM as the main model in the remaining experiments. Second, scaling MDM with the MoE Transformer architecture improves compute efficiency over the dense counterpart by $1.36\times$, and allocating compute according to the optimal N_t^* and D_t^* derived from Ermine yields a further $1.35\times$ improvement. Combined, the two techniques narrow the compute gap between MDM and ARM from $17.56\times$ to $9.57\times$. Third, the efficiency gain from the Ermine allocation ($1.35\times$) matches that of the MoE architecture ($1.36\times$), indicating that reallocating compute across time is as effective as a $3\times$ expansion in total parameters.

4.2 Ermine Improves Likelihood Estimation

We evaluate our models on OWT (Gokaslan et al., 2019), the benchmark commonly adopted in the DLM literature. Beyond predicting optimal allocations, the fitted scaling laws serve as a diagnostic tool for examining whether an existing training recipe is well-configured. The setup used in prior studies (i.e., N=92M (non-embedding) parameters, D=524B tokens) deviates substantially from the compute-optimal frontier. At the same budget of 2.89×10^{20} FLOPs, the laws instead prescribe a considerably larger model trained on far fewer tokens (see Fig. A3).

Motivated by this observation, we first adjust each method to its compute-optimal configuration according to Chinchilla and Ermine, and then compare their performance under the identical total budget. Table 2 summarizes the results. Switching to the compute-optimal setup yields significant improvements for ARM (17.54 → 12.98), MDM (22.98 → 18.85), and MDM-MoE (21.44 → 18.15), validating the effectiveness of scaling-law-guided training. Moreover, allocating compute with the Ermine scaling law offers a further noticeable improvement. The semi-time-dependent variants (Time-invariant D and Time-invariant N) achieve perplexities of 17.72 and 17.70, and the standard Ermine allocation attains 17.63, outperforming all existing order-agnostic baselines.

4.3 Ermine Accelerates Large-Scale Pre-training

To test the time-dependent allocation at scale, we train MDM-MoE for 1.1×10^{21} FLOPs on the Dolma-1.7 pretraining corpus (Soldaini et al., 2024), comparing the Ermine and Chinchilla allocations. We report zero-shot accuracy on five commonsense reasoning benchmarks (SciQ, SIQA, ARC-e, HellaSwag, and PIQA. See Table A8 for detailed description). The validation loss and zero-shot accuracy over the course of training are shown in Figs. A5 and A4.

Table 3: Zero-shot accuracies on reasoning benchmarks. **Bold** marks the best value among models in the first two blocks. Models in the third block use $\geq 10\times$ FLOPs and are included for **reference**.

	FLOPs	SciQ	SIQA	ARC-e	HellaS	PIQA	Avg.
Aligned Models (identical backbone and data)							
MDM-MoE (Chinchilla; A1.1B)	1.1×10^{21}	82.00	41.86	49.64	48.49	63.22	57.04
MDM-MoE (Ermine; A1.1B)	1.1×10^{21}	**86.50**	**42.47**	**56.14**	48.53	66.43	**60.01**
Reference Models ($\sim$2-3$\times$ FLOPs)							
GPT-Neo (1.3B) (Black et al., 2022)	3.1×10^{21}	77.10	41.25	50.21	48.93	71.06	57.71
OPT (1.3B) (Zhang et al., 2022)	2.4×10^{21}	76.70	40.63	50.97	53.70	**72.47**	58.89
Pythia (1.4B) (Biderman et al., 2023)	2.5×10^{21}	79.20	40.94	53.91	52.07	71.00	59.42
Bloom (1.1B) (BigScience, 2023)	2.2×10^{21}	74.60	39.05	45.41	42.96	67.19	53.84
TinyLLaMA (1.1B) (Zhang et al., 2024)	3.3×10^{21}	80.90	39.56	50.13	**54.11**	70.51	59.04
SMDM (1.1B) (Nie et al., 2025)	3.3×10^{21}	81.20	41.04	46.13	51.83	69.53	57.95
MDM-Prime-v2 (1.1B) (Chao et al., 2026)	3.3×10^{21}	83.30	42.02	47.81	48.50	62.57	56.84
Large-Compute Reference Models ($\geq 10\times$ FLOPs)							
Gemma3 (1.0B) (Gemma Team, 2025)	1.2×10^{22}	90.50	41.76	69.02	57.72	72.58	66.32
OLMoE (A1.3B) (Muennighoff et al., 2025)	3.0×10^{22}	94.80	43.71	78.07	76.97	80.36	74.78

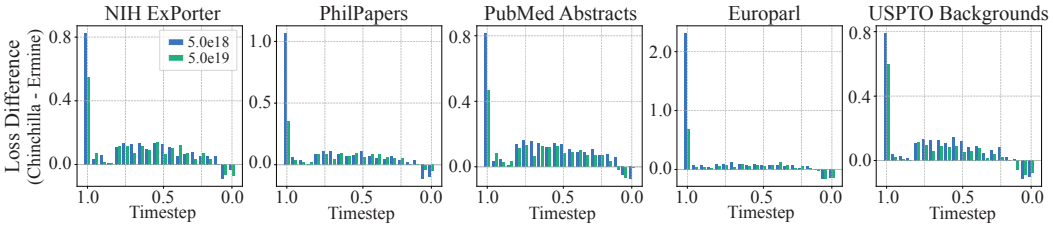

Figure 8: Time-wise zero-shot loss difference (Chinchilla − Ermine) for MDM-MoE on held-out Pile domains at $C = 5 \times 10^{18}$ and 5×10^{19} FLOPs. Positive bars indicate Ermine attains lower loss.

Table 3 summarizes the final accuracy. Ermine improves over Chinchilla on all five benchmarks and by 2.97 points on average, with the largest gains on ARC-e (+6.50) and SciQ (+4.50). Relative to external baselines trained at 2-3$\times$ our budget, Ermine attains the highest average accuracy (60.01 against 59.42 for the strongest baseline) and the best individual scores on SciQ, SIQA, and ARC-e. The large-compute reference models (Gemma3, OLMoE) indicate the accuracy reachable at 10-30$\times$ the compute and are included as an upper reference.

4.4 Generalization Beyond Chinchilla

A DLM's scalability is not uniform across diffusion timesteps, and neither should its generalizability be. Section 3.1 (Fig. 4 **(5)**) shows that γ_t decreases with t, i.e., the noisy end scales worse, and C_t^* accordingly prescribes less compute there than a uniform schedule does. A uniform (Chinchilla) allocation over-trains these regions relative to what they can support. The in-distribution time-wise validation loss increases during training in the noisy time regions (Fig. A6). To test whether this over-training also degrades generalization behavior, we evaluate MDM-MoE trained under Chinchilla and Ermine on held-out textual domains from the Pile (Gao et al., 2020) and compare their time-wise zero-shot validation losses. Fig. 8 reports the difference, where positive values indicate that Ermine generalizes better at that timestep. The gap is largest as $t \to 1$, matching the region where γ_t is lowest. The result thus suggests that Ermine improves not only training efficiency but also zero-shot generalization, by allocating to each time region the compute it can absorb.

5 Conclusion

This paper identifies a time-wise non-uniformity in DLM scaling, where loss improves unevenly across diffusion timesteps, leaving compute underutilized under Chinchilla-style allocation. We introduce Ermine, a scaling law that optimally prescribes time-dependent allocation of model capacity and data. Ermine establishes an improved efficiency frontier, accelerates large-scale pre-training, and generalizes better by avoiding excess compute where scalability diminishes.

AI USE STATEMENT

In this work, we used generative AI tools for grammar and language editing of the manuscript. All AI-suggested edits to the text were checked by the authors to confirm that the intended meaning and technical claims were preserved. Additionally, we used generative AI tools for writing the Python code that produces the figures in this paper, and for resolving software dependency issues when configuring the environments used to run our experiments. We have not used generative AI tools for research ideation, for the design of our method or experiments, for deriving or proving any theoretical results, or for generating any results, data, or analyses reported in this work. We have reviewed all AI-assisted work. All AI-generated code was read, executed, and verified for correctness by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The complete experimental configuration, including model architectures and sizes, optimizer settings, learning rate schedules, and all other hyperparameters, is documented in Appendix A.5. An anonymized, downloadable link to our source code, with instructions for reproducing the experiments, is provided in Appendix A.6. To allow full transparency, we will also release the training logs (Figs. 7, A4, A5, and A6) and the pre-trained model weights (Tables 2, 3, and A11) corresponding to most result reported in this paper. For our theoretical contributions, the assumptions, formal statements, and complete proofs of all claims are given in Appendices A.1 and A.2.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2023.

BigScience. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. `arXiv:2211.05100 [cs.CL]`, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about Physical Commonsense in Natural Language. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. `arXiv:2204.06745 [cs.CL]`, 2022.

Chen-Hao Chao, Wei-Fang Sun, Hanwen Liang, Chun-Yi Lee, and Rahul G. Krishnan. Beyond Masked and Unmasked: Discrete Diffusion Models via Partial Masking. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Chen-Hao Chao, Wei-Fang Sun, Junwei Quan, Chun-Yi Lee, and Rahul G. Krishnan. MDM-Prime-v2: Binary Encoding and Index Shuffling Enable Scaling of Diffusion Language Models. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Philipp Koehn, and Tony Robinson. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. In *Proc. Conf. of the European Chapter of the Association for Computational Linguistics (EACL)*, 2013.

Kyle Chickering, Wei-An Lin, Swayam Bhanded, Dan Saunders, Akshat Tripathi, Jiaming Song, Shyamal Buch, and Xinchen Yan. Abra: Scaling Diffusion Image Training. arXiv:2608.17286 [cs.LG], 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. 2018.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. In *Proc. Conf. of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd Van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling Vision Transformers to 22 Billion Parameters. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2023.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling Rectified Flow Transformers for High-Resolution Image Synthesis. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2024.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. *Proc. Machine Learning and Systems*, 5, 2023.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. arXiv:2101.00027 [cs.CL], 2020.

Gemma Team. Gemma 3 Technical Report. arXiv:2503.19786 [cs.CL], 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2023.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. 2019.

Tiankai Hang, Shuyang Gu, Xin Geng, and Baining Guo. Improved Noise Schedule for Diffusion Training. In *Proc. Int. Conf. on Computer Vision, (ICCV)*, 2024.

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:1712.00409 [cs.LG], 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

Chunsan Hong, Sanghyun Lee, Chieh-Hsin Lai, Satoshi Hayakawa, Yuhta Takida, Yuki Mitsufuji, Seungryong Kim, and Jong Chul Ye. Understanding and accelerating the training of masked diffusion language models. `arXiv:2605.13026 [cs.LG]`, 2026.

Peter J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35: 492–518, 1964.

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:2001.08361 [cs.LG]`, 2020.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

Autar K Kaw, Egwu K Kalu, and Duc Nguyen. *Numerical Methods with Applications*. 2009.

Diederik P. Kingma and Ruiqi Gao. Understanding Diffusion Objectives as the ELBO with Simple Data Augmentation. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2023.

Zhengyang Liang, Hao He, Ceyuan Yang, and Bo Dai. Scaling Laws For Diffusion Transformers. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow Matching for Generative Modeling. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2019.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2024.

Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, and Sebastian Jaszczur. Joint MoE Scaling Laws: Mixture of Experts Can Be Memory Efficient. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2025.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics (CL)*, pp. 313–330, 1993.

Kangfu Mei, Zhengzhong Tu, Mauricio Delbracio, Hossein Talebi, Vishal M. Patel, and Peyman Milanfar. Bigger is not Always Better: Scaling Properties of Latent Diffusion Models. *Transactions on Machine Learning Research (TMLR)*, 2025. ISSN 2835-8856.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. `arXiv:1609.07843 [cs.CL]`, 2016.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling Data-Constrained Language Models. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2023.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Evan Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open Mixture-of-Experts Language Models. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2025.

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:2510.03280 [cs.LG]`, 2025.

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 *Proc. Int. Conf. on Learning Representations (ICLR)*, 2025.

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

Tim Pearce and Jinyeop Song. Reconciling Kaplan and Chinchilla Scaling Laws. *Transactions on Machine Learning Research (TMLR)*, 2024. ISSN 2835-8856.

William Peebles and Saining Xie. Scalable Diffusion Models with Transformers. In *Proc. Int. Conf. on Computer Vision, (ICCV)*, 2023.

Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion Beats Autoregressive in Data-Constrained Settings. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2025.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research (JMLR)*, 21(1), 2020. ISSN 1532-4435.

Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A Constructive Prediction of the Generalization Error Across Scales. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2020.

Dimitri von Rütte, Janis Fluri, Omead Pooladzandi, Bernhard Schölkopf, Thomas Hofmann, and Antonio Orvieto. Scaling Behavior of Discrete Diffusion Language Models. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2026.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The Diffusion Duality. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2025.

Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukić. Scaling Beyond Masked Diffusion Language Models. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2026a.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric Language Models: A Family of Any-Order Diffusion LLMs. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2026b.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense Reasoning about Social Interactions. In *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-torre, Bernardo P. de Almeida, Alexander Rush, Thomas Pierrot, and Volodymyr Kuleshov. Simple Guidance Mechanisms for Discrete Diffusion Models. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2025.

Ivan Sedykh, Nikita Sorokin, and Valentin Malykh. Not All Denoising Steps Are Equal: Model Scheduling for Faster Masked Diffusion Language Models. arXiv:2604.02340 [cs.LG], 2026.

Noam Shazeer. GLU Variants Improve Transformer. arXiv:2002.05202 [cs.LG], 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. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2017.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and Generalized Masked Diffusion for Discrete Data. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL], 2019.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2021.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing*, 2024. ISSN 0925-2312.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL], 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Proc. of Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2017.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized Interpolating Discrete Diffusion. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2025.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Yuanyang Yin, Yaqi Zhao, Mingwu Zheng, Ke Lin, Jiarong Ou, Rui Chen, Victor Shea-Jay Huang, Jiahao Wang, Xin Tao, Pengfei Wan, Di Zhang, Baoqun Yin, Wentao Zhang, and Kun Gai. Towards Precise Scaling Laws for Video Diffusion Transformers. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An Open-Source Small Language Model. arXiv:2401.02385 [cs.CL], 2024.

Shuibai Zhang, Caspian Zhuang, Chihan Cui, Zhihan Yang, Fred Zhangzhi Peng, Yanxin Zhang, Haoyue Bai, Zack Jia, Yang Zhou, Guanhua Chen, and Ming Liu. Expert-Choice Routing Enables Adaptive Computation in Diffusion Language Models. In *Proc. Conf. on Language Modeling (COLM)*, 2026.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL], 2022.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. In *Proc. of Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2015.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam M. Shazeer, and William Fedus. ST-MoE: Designing Stable and Transferable Sparse Expert Models. arXiv:2202.08906 [cs.CL], 2022.

A APPENDIX

This appendix provides additional explanations, analyses, and experiments. Section A.1 presents the theoretical derivation of Ermine. Section A.2 offers detailed formulations of existing diffusion language models (DLM). Section A.3 lists and compares a number of related works. Section A.4 outlines some technical details of the proposed method. Section A.5 details the experimental configurations. Section A.6 reports additional experimental results. Section A.7 summarizes the limitations of this work.

A.1 THEORETICAL ANALYSIS OF ERMINE

This section provides theoretical derivation of the Ermine scaling laws. A list of notations used in Chinchilla and Ermine is presented in Table A1. For easy reference, we move Eq. (9) here, which serves as the main equation in the theoretical derivation:

$$\mathcal{J}(N_t, D_t, \mu) = \int_0^1 \left(E_t + A_t N_t^{-a_t} + B_t D_t^{-b_t}\right) dt + \mu \left(\int_0^1 6 N_t D_t dt - C\right). \tag{A1}$$

Table A1: Notation correspondence between Chinchilla and Ermine.

Chinchilla	Ermine	Definition
N	N_t	Model capacity: total parameter count in Chinchilla; parameter density at time t in Ermine.
D	D_t	Training tokens: total token count in Chinchilla; data density at time t in Ermine.
E	E_t	Irreducible loss: the loss floor no amount of compute can improve on.
a	a_t	Capacity exponent: slope of the capacity penalty in log space.
A	A_t	Capacity amplitude: shift of the capacity penalty in log space.
b	b_t	Data exponent: slope of the data penalty in log space.
B	B_t	Data amplitude: shift of the data penalty in log space.
G	G_t	Split ratio: $\sqrt{N^*/D^*}$ when the data and capacity exponents are equal. (form: $(\frac{aA}{bB})^{\frac{1}{a+b}}$)
γ	γ_t	Compute exponent: slope of the compute-loss frontier in log space. (form: $\frac{ab}{a+b}$)
F	F_t	Compute amplitude: shift of the compute-loss frontier in log space. (form: $6^{\gamma}(AG^{-a} + BG^{b})$)
–	μ	Lagrange multiplier: use to solve the constrained optimization problem (Eq. (8)).

A.1.1 DERIVATION OF PROPOSITION 1

This section presents the theoretical rationale behind Ermine. We first establish two lemmas (**Lemmas** A1 and A2), and then build on them to derive **Proposition** 1.

Lemma A1. *Let* $C_t \triangleq 6N_tD_t$ *and* $G_t \triangleq \left(\frac{a_tA_t}{b_tB_t}\right)^{\frac{1}{a_t+b_t}}$. *The optimal solution to Eq. (A1) satisfies:*

$$N_t^* = G_t \left(\frac{C_t}{6}\right)^{\frac{b_t}{a_t+b_t}}, \quad D_t^* = G_t^{-1} \left(\frac{C_t}{6}\right)^{\frac{a_t}{a_t+b_t}}. \tag{A2}$$

Proof. Taking pointwise first-order conditions of $\mathcal{J}$ yields:

$$\frac{\partial \mathcal{J}}{\partial N_t} = -a_t A_t N_t^{-a_t-1} + 6\mu D_t = 0, \qquad \frac{\partial \mathcal{J}}{\partial D_t} = -b_t B_t D_t^{-b_t-1} + 6\mu N_t = 0.$$

We define N_t^* and D_t^* as the variables that satisfy the above conditions. By rearranging the first-order conditions, we have:

$$\begin{aligned}
&a_tA_t(N_t^*)^{-a_t-1} = 6\mu D_t^*, \quad b_tB_t(D_t^*)^{-b_t-1} = 6\mu N_t^*.\\
&\Rightarrow a_tA_t(N_t^*)^{-a_t} = 6\mu N_t^* D_t^*, \quad b_tB_t(D_t^*)^{-b_t} = 6\mu N_t^* D_t^*.\\
&\Rightarrow a_tA_t(N_t^*)^{-a_t} = b_tB_t(D_t^*)^{-b_t}.\\
&\Rightarrow (D_t^*)^{b_t} = \frac{b_tB_t}{a_tA_t}(N_t^*)^{a_t}. \quad \Rightarrow (D_t^*) = \left(\frac{b_tB_t}{a_tA_t}\right)^{1/b_t} (N_t^*)^{a_t/b_t}.
\end{aligned}$$

Note that μ cancels, so the split depends only on C_t. We then apply the formula $C_t = 6N_t^* D_t^*$ to express N_t^* and D_t^* as functions of C_t:

$$\Rightarrow C_t = 6N_t^* D_t^* = 6N_t^* \left(\frac{b_t B_t}{a_t A_t}\right)^{1/b_t} (N_t^*)^{a_t/b_t} = 6\left(\frac{b_t B_t}{a_t A_t}\right)^{1/b_t} (N_t^*)^{(a_t+b_t)/b_t}.$$

$$\Rightarrow (N_t^*)^{\frac{a_t+b_t}{b_t}} = \frac{C_t}{6}\left(\frac{a_t A_t}{b_t B_t}\right)^{1/b_t}.$$

$$\Rightarrow N_t^* = \left[\frac{C_t}{6}\left(\frac{a_t A_t}{b_t B_t}\right)^{1/b_t}\right]^{\frac{b_t}{a_t+b_t}} = \left(\frac{a_t A_t}{b_t B_t}\right)^{\frac{1}{a_t+b_t}} \left(\frac{C_t}{6}\right)^{\frac{b_t}{a_t+b_t}} = G_t \left(\frac{C_t}{6}\right)^{\frac{b_t}{a_t+b_t}}.$$

$$\Rightarrow D_t^* = \frac{C_t}{6N_t^*} = \frac{C_t}{6}\left[G_t\left(\frac{C_t}{6}\right)^{\frac{b_t}{a_t+b_t}}\right]^{-1} = G_t^{-1}\left(\frac{C_t}{6}\right)^{1-\frac{b_t}{a_t+b_t}} = G_t^{-1}\left(\frac{C_t}{6}\right)^{\frac{a_t}{a_t+b_t}}.$$

□

Lemma A2. *Let* $\gamma_t \triangleq \frac{a_t b_t}{a_t+b_t}$, *and* $F_t \triangleq 6^{\gamma_t}(A_t G_t^{-a_t} + B_t G_t^{b_t})$. *The optimal time-wise loss* $\hat{\ell}_t(N_t^*, D_t^*)$ *takes the following form:*

$$\hat{\ell}_t(N_t^*, D_t^*) = E_t + F_t C_t^{-\gamma_t}. \tag{A3}$$

Proof. Substituting N_t^* and D_t^* in **Lemma** A1 into the time-wise loss gives:

$$\begin{aligned}
\hat{\ell}_t(N_t^*, D_t^*) &= E_t + A_t (N_t^*)^{-a_t} + B_t (D_t^*)^{-b_t} \\
&= E_t + A_t \left[G_t \left(\frac{C_t}{6}\right)^{\frac{b_t}{a_t+b_t}}\right]^{-a_t} + B_t \left[G_t^{-1}\left(\frac{C_t}{6}\right)^{\frac{a_t}{a_t+b_t}}\right]^{-b_t} \\
&= E_t + A_t G_t^{-a_t}\left(\frac{C_t}{6}\right)^{-\frac{a_t b_t}{a_t+b_t}} + B_t G_t^{b_t}\left(\frac{C_t}{6}\right)^{-\frac{a_t b_t}{a_t+b_t}} \\
&= E_t + \left[A_t G_t^{-a_t} + B_t G_t^{b_t}\right]\left(\frac{C_t}{6}\right)^{-\frac{a_t b_t}{a_t+b_t}} \\
&= E_t + F_t C_t^{-\gamma_t}.
\end{aligned}$$

□

Proposition 1. *Given a target compute budget* C, *the optimal solution to Eq. (A1) is:*

$$C_t^* = \left(\frac{\gamma_t F_t}{\mu^*}\right)^{\frac{1}{1+\gamma_t}}, \quad N_t^* = G_t\left(\frac{C_t^*}{6}\right)^{\frac{b_t}{a_t+b_t}}, \quad D_t^* = G_t^{-1}\left(\frac{C_t^*}{6}\right)^{\frac{a_t}{a_t+b_t}}, \tag{A4}$$

where μ^* *is unique and satisfies* $\int_0^1 (\gamma_t F_t/\mu)^{\frac{1}{1+\gamma_t}}\, dt = C$.

Proof. Applying the result of **Lemma** A2, the Lagrangian $\mathcal{J}(N_t^*, D_t^*, \mu)$ can be written as a function of C_t as follows:

$$\begin{aligned}
\mathcal{J}(N_t^*, D_t^*, \mu) &= \int_0^1 \left(E_t + A_t (N_t^*)^{-a_t} + B_t (D_t^*)^{-b_t}\right) dt + \mu\left(\int_0^1 6N_t^* D_t^* dt - C\right) \\
&= \int_0^1 \left(E_t + F_t C_t^{-\gamma_t}\right) dt + \mu\left(\int_0^1 C_t\, dt - C\right) \\
&= \int_0^1 \left(E_t + F_t C_t^{-\gamma_t} + \mu C_t\right) dt - \mu C.
\end{aligned}$$

Taking the pointwise first-order condition with respect to C_t yields:

$$\frac{\partial \mathcal{J}}{\partial C_t} = -\gamma_t F_t C_t^{-\gamma_t - 1} + \mu = 0.$$

By rearranging the first-order condition, we have:

$$\mu = \gamma_t F_t C_t^{-\gamma_t - 1}. \quad \Rightarrow \quad C_t^{\gamma_t+1} = \frac{\gamma_t F_t}{\mu}. \quad \Rightarrow \quad C_t = \left(\frac{\gamma_t F_t}{\mu}\right)^{\frac{1}{1+\gamma_t}}.$$

The multiplier is then determined by the first-order condition with respect to μ, which recovers the budget constraint:

$$\frac{\partial \mathcal{J}}{\partial \mu} = \int_0^1 C_t \, dt - C = 0. \quad \Rightarrow \quad \int_0^1 \left(\frac{\gamma_t F_t}{\mu}\right)^{\frac{1}{1+\gamma_t}} dt = C.$$

Denoting by μ^* the solution to the above equation, C_t^* is the variable satisfying both first-order conditions, i.e., $C_t^* = (\gamma_t F_t/\mu^*)^{\frac{1}{1+\gamma_t}}$, which recovers the optimal form in Eq. (A4).

Regarding the uniqueness of μ^*, let $\Phi(\mu) \triangleq \int_0^1 (\gamma_t F_t/\mu)^{\frac{1}{1+\gamma_t}} dt$. $\Phi(\mu)$ is continuous and strictly decreasing in μ, with $\Phi(\mu) \to \infty$ as $\mu \to 0^+$ and $\Phi(\mu) \to 0$ as $\mu \to \infty$. As a result, a unique μ^* exists for any $C > 0$. □

A.1.2 Derivation of Time-Invariant Allocation

Sections A.1.1 and 3.1 describe the training setup where both N_t and D_t vary with t. This section discusses semi-time-dependent setups where only one of them does, i.e., either $N_t = N$ or $D_t = D$, $\forall t \in [0, 1]$, with the time-invariant variable still optimized as a single value shared across all timesteps. The analytic solutions for ***(i)*** Time-Invariant N and ***(ii)*** Time-Invariant D are given in **Proposition** A1.

Proposition A1. *Given C, define $\tilde{\mu}(N) \triangleq \frac{6N}{C}\int_0^1 a_t A_t N^{-a_t} dt$ and $\tilde{\mu}(D) \triangleq \frac{6D}{C}\int_0^1 b_t B_t D^{-b_t} dt$. The optimal solutions to Eq. (A1) when one variable is held time-invariant have the following forms:*

$$\textit{(i) Time-Invariant } N\textit{:}\ D_t^* = \left(\frac{b_t B_t}{\tilde{\mu}(N^*)}\right)^{\frac{1}{1+b_t}}, \qquad \textit{(ii) Time-Invariant } D\textit{:}\ N_t^* = \left(\frac{a_t A_t}{\tilde{\mu}(D^*)}\right)^{\frac{1}{1+a_t}}, \tag{A5}$$

*where N^* in case **(i)** is the solution to $6N\int_0^1 (b_t B_t/\tilde{\mu}(N))^{\frac{1}{1+b_t}}\ dt = C$ and D^* in case **(ii)** is the solution to $6D\int_0^1 (a_t A_t/\tilde{\mu}(D))^{\frac{1}{1+a_t}}\ dt = C$. The solutions N^* and D^* are unique.*

Proof. We prove case ***(i)***; case ***(ii)*** follows by exchanging the roles of (N, a_t, A_t) and (D, b_t, B_t). With N held time-invariant, the Lagrangian in Eq. (A1) reads

$$\mathcal{J}(N, D_t, \mu) = \int_0^1 \left(E_t + A_t N^{-a_t} + B_t D_t^{-b_t}\right) dt + \mu\left(\int_0^1 6N D_t \, dt - C\right).$$

Taking the pointwise first-order condition with respect to D_t yields:

$$\frac{\partial \mathcal{J}}{\partial D_t} = -b_t B_t D_t^{-b_t - 1} + 6\mu N = 0 \quad \Rightarrow \quad D_t = \left(\frac{b_t B_t}{6\mu N}\right)^{\frac{1}{1+b_t}}.$$

The first-order condition with respect to N couples all timesteps:

$$\frac{\partial \mathcal{J}}{\partial N} = -\int_0^1 a_t A_t N^{-a_t - 1} dt + 6\mu \int_0^1 D_t \, dt = 0 \quad \Rightarrow \quad \int_0^1 a_t A_t N^{-a_t} dt = 6\mu N \int_0^1 D_t \, dt,$$

where the second expression follows by multiplying both sides by N. Substituting the budget constraint $6N\int_0^1 D_t \, dt = C$ into the right-hand side yields $\int_0^1 a_t A_t N^{-a_t} dt = \mu C$, and hence

$$6\mu N = \frac{6N}{C}\int_0^1 a_t A_t N^{-a_t} dt = \tilde{\mu}(N).$$

Let N^* be the model size satisfying the budget constraint. Substituting $6\mu N^* = \tilde{\mu}(N^*)$ into the expression for D_t, we obtain $D_t^* = (b_t B_t/\tilde{\mu}(N^*))^{\frac{1}{1+b_t}}$, which recovers Eq. (A5).

Regarding the uniqueness of N^*, let $\mathcal{A}(N) \triangleq \int_0^1 a_t A_t N^{-a_t} dt$, so that $\tilde{\mu}(N) = 6N\mathcal{A}(N)/C$ and

$$\Psi(N) \triangleq 6N \int_0^1 \left(\frac{b_t B_t}{\tilde{\mu}(N)}\right)^{\frac{1}{1+b_t}} dt = \int_0^1 (6N)^{\frac{b_t}{1+b_t}} \left(\frac{b_t B_t\, C}{\mathcal{A}(N)}\right)^{\frac{1}{1+b_t}} dt.$$

Since $b_t > 0$, the factor $(6N)^{\frac{b_t}{1+b_t}}$ is strictly increasing in N; since $a_t > 0$, $\mathcal{A}(N)$ is strictly decreasing in N and hence the second factor is strictly increasing as well. Since both factors are positive and increase from 0 to ∞, $\Psi(N)$ is continuous and strictly increasing in N, with $\Psi(N) \to 0$ as $N \to 0^+$ and $\Psi(N) \to \infty$ as $N \to \infty$. As a result, a unique N^* attains any target $C > 0$. □

In practice, Eq. (A5) is solved with the bisection method, similar to the standard Ermine. Due to the asymmetry of the time-invariant cases, the Lagrange multiplier μ is eliminated and replaced by the capacity- (or data-) dependent quantities $\tilde{\mu}(N^*)$ and $\tilde{\mu}(D^*)$ (see Figs. A12 and A13).

D_t^* and N_t^* in Eq. (A5) share the same form as C_t^* in Eq. (10), respectively. Consider case ***(i)***. With N fixed, its penalty can be absorbed into the irreducible term and the time-wise loss becomes $\hat{\ell}_t(N, D_t) = (E_t + A_t N^{-a_t}) + B_t D_t^{-b_t}$. The product $b_t B_t$ in the numerator of D_t^* is the amplitude of the marginal-return $-d\hat{\ell}_t/dD_t$, which Eq. (A5) balances against $\tilde{\mu}(N^*)$; the latter acts as a capacity-dependent temperature that flattens or sharpens D_t^*. The analogy also applies to case ***(ii)***.

A.2 Detailed Formulations of DLMs

Section 2 discusses how DLMs are formulated. This section provides the proof for Eq. (7) (**Proposition** A2) and the detailed formulations of MDMs and UDMs, which are two widely-adopted DLM families. In the following formulations, we denote the parameterized denoiser, a softmax distribution with V slots, that capture the x_0^i distribution as $p_\theta(x_0^i|\boldsymbol{x}_t)$. A list of notations used in the DLM formulations is presented in Table A2.

Table A2: Notations used in DLM formulations.

Symbol	Definition	Symbol	Definition
L	Token sequence length	$\delta_x(x')$	Kronecker delta (equals 1 only if $x = x'$)
V	Number of classes a token can take	$\mathcal{X}$	A set of tokens: $\{1, \cdots, V\}$
α_t	Scheduling function ($\alpha_t : [0,1] \to [0,1]$)	$\boldsymbol{x}_0$	A sequence of clean tokens ($\boldsymbol{x}_0 \in \mathcal{X}^L$)
$\alpha_{t\|s}$	Ratio of two scheduling functions ($\alpha_{t\|s} = \alpha_t/\alpha_s$)	$\boldsymbol{x}_t$	A sequence of noised tokens ($\boldsymbol{x}_t \in \mathcal{X}^L$)
π	The prior distribution	x_0^i	The i-th element of $\boldsymbol{x}_0$
q	The forward / transition diffusion kernel	x_t^i	The i-th element of $\boldsymbol{x}_t$
p_θ	The reversed kernel parameterized by θ	$\mathtt{m}$	A special masked token

Proposition A2. *The time-wise loss in Eq. (5) can be decomposed as an irreducible and reducible terms:*

$$\begin{aligned}&\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]\\ &= \underbrace{\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,q(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]}_{\textit{irreducible}} + \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t)\,\|\,p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]}_{\textit{reducible}}.\end{aligned} \tag{A6}$$

Proof.

$$
\begin{aligned}
&\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big] \\
&= \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\sum_{\boldsymbol{x}_s} q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\left[\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)}{p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)}\right]\right] \\
&\overset{(i)}{=} \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\sum_{\boldsymbol{x}_s} q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\left[\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)}{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}+\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}{p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)}\right]\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\sum_{\boldsymbol{x}_s} q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\left[\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)}{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}\right]\right]+\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\sum_{\boldsymbol{x}_s} q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}{p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)}\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,q(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]+\sum_{\boldsymbol{x}_t}\sum_{\boldsymbol{x}_0} q(\boldsymbol{x}_t)q(\boldsymbol{x}_0|\boldsymbol{x}_t)\sum_{\boldsymbol{x}_s} q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}{p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)} \\
&\overset{(ii)}{=} \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,q(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]+\sum_{\boldsymbol{x}_t} q(\boldsymbol{x}_t)\sum_{\boldsymbol{x}_s}\left[\sum_{\boldsymbol{x}_0} q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)q(\boldsymbol{x}_0|\boldsymbol{x}_t)\right]\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}{p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)} \\
&= \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,q(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]+\sum_{\boldsymbol{x}_t} q(\boldsymbol{x}_t)\sum_{\boldsymbol{x}_s} q(\boldsymbol{x}_s|\boldsymbol{x}_t)\log\frac{q(\boldsymbol{x}_s|\boldsymbol{x}_t)}{p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)} \\
&= \mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t,\boldsymbol{x}_0)\,\|\,q(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big]+\mathbb{E}_{q(\boldsymbol{x}_t)}\Big[D_{\mathrm{KL}}\big[q(\boldsymbol{x}_s|\boldsymbol{x}_t)\,\|\,p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t)\big]\Big],
\end{aligned}
$$

where step (i) leverages the formula $\log\frac{X}{Y}=\log X-\log Y=\log X-\log Z+\log Z-\log Y=\log\frac{X}{Z}+\log\frac{Z}{Y}$, and step (ii) reorders the summation of $\boldsymbol{x}_0$ and $q(\boldsymbol{x}_0|\boldsymbol{x}_t)$. □

For MDMs, the prior is defined as $\pi(x_1^i)=\delta_{\mathtt{m}}(x_1^i)$ with a special masked symbol $\mathtt{m}$. The time-wise loss ℓ_t^{MDM} is a simple cross-entropy loss on masked tokens:

$$
\ell_t^{\mathrm{MDM}}=\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\frac{\alpha_t'}{1-\alpha_t}\sum_{i=1}^{L}\delta_{\mathtt{m}}(x_t^i)\log p_\theta(x_0^i\mid\boldsymbol{x}_t)\right], \tag{A7}
$$

where $\alpha_t'=\frac{d\alpha_t}{dt}$ is the time derivative of the scheduler.

For UDMs, the prior is defined as $\pi(x_1^i)=\frac{1}{V}$. Let $\bar{p}_\theta^i(v)\triangleq\alpha_t\,p_\theta(x_0^i=v|\boldsymbol{x}_t)+(1-\alpha_t)\frac{1}{V}$ denote the model prediction diffused by the forward kernel. In addition, let $\bar{p}^i(v)\triangleq\alpha_t\,\delta_{x_0^i}(v)+(1-\alpha_t)\frac{1}{V}$ be its corresponding ground truth distribution. The time-wise loss ℓ_t^{UDM} is written as follows (Schiff et al., 2025):

$$
\ell_t^{\mathrm{UDM}}=\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\frac{\alpha_t'}{V\alpha_t}\sum_{i=1}^{L}\left(\frac{1}{\bar{p}^i(x_t^i)}+\frac{1}{\bar{p}_\theta^i(x_t^i)}+\sum_{v=1}^{V}(1-\delta_{x_t^i}(v))\frac{\bar{p}^i(v)}{\bar{p}^i(x_t^i)}\log\frac{\bar{p}_\theta^i(x_t^i)\bar{p}^i(v)}{\bar{p}_\theta^i(v)\bar{p}^i(x_t^i)}\right)\right], \tag{A8}
$$

To achieve a lower-variance training, Sahoo et al. (2025) proposed an equivalent variant of Eq. (A8), called Duo. Define constants $\kappa_t\triangleq\frac{1-\alpha_t}{V\alpha_t+1-\alpha_t}$ and $\beta_t^i\triangleq\delta_{x_0^i}(x_t^i)\kappa_t+\Big(1-\delta_{x_0^i}(x_t^i)\Big)$. The time-wise loss can be decomposed as four terms $\tilde{\ell}_t^1,\tilde{\ell}_t^2,\tilde{\ell}_t^3,\tilde{\ell}_t^4$ as follows:

$$
\ell_t^{\mathrm{UDM}}=\mathbb{E}_{q(\boldsymbol{x}_0,\boldsymbol{x}_t)}\left[\frac{\alpha_t'}{V\alpha_t}\sum_{i=1}^{L}\Big(\tilde{\ell}_t^1+\tilde{\ell}_t^2+\tilde{\ell}_t^3+\tilde{\ell}_t^4\Big)\right], \tag{A9}
$$

where $\tilde{\ell}_t^1,\tilde{\ell}_t^2,\tilde{\ell}_t^3,\tilde{\ell}_t^4$ are respectively defined as:

$$
\begin{aligned}
\tilde{\ell}_t^1&=\frac{1}{\bar{p}^i(x_t^i)}+\frac{1}{\bar{p}_\theta^i(x_t^i)}, &\qquad \tilde{\ell}_t^3&=-V\frac{\alpha_t}{1-\alpha_t}\Big(1-\delta_{x_0^i}(x_t^i)\Big)\log\frac{\bar{p}_\theta^i(x_t^i)}{\bar{p}_\theta^i(x_0^i)}, \\
\tilde{\ell}_t^2&=-\beta_t^i\sum_{v=1}^{V}\log\frac{\bar{p}_\theta^i(x_t^i)}{\bar{p}_\theta^i(v)}, &\qquad \tilde{\ell}_t^4&=-\left((V-1)\delta_{x_0^i}(x_t^i)\kappa_t-\frac{1}{\kappa_t}\Big(1-\delta_{x_0^i}(x_t^i)\Big)\right)\log\kappa_t.
\end{aligned} \tag{A10}
$$

In our experiments, we adopt Eq. (A9).

Table A3: Comparison of time-dependent compute allocation methods. Ermine differs in three respects. (1) *Mechanism*: the schedule is learned by fitting a time-dependent scaling law, rather than imposed as a distribution family whose hyperparameters requires grid-searched (last column). (2) *Variable*: N_t and D_t are allocated jointly, with the constant-N and constant-D cases recovered as special cases, whereas prior work tunes a single axis. (3) C*-dependency*: the Ermine allocation changes across the target budget C, while prior schedules are fixed across budgets.

	Allocation Mechanism	Variable	C-dependency	Hyperparam.-free
Ermine (ours)	**Learned**: fitted scaling law	(N_t, D_t)/N_t/D_t	✓	✓
Karras et al. (2022)	**Inductive bias**: Gaussian	D_t	✗	✗
Kingma & Gao (2023)	**Inductive bias**: Gaussian / cosine	D_t	✗	✗
Esser et al. (2024)	**Inductive bias**: Gaussian	D_t	✗	✗
Hang et al. (2024)	**Inductive bias**: Gaus. / Lap. / Cauchy / cosine	D_t	✗	✗
Arriola et al. (2025)	**Inductive bias**: truncated uniform	D_t	✗	✗
Hong et al. (2026)	**Inductive bias**: Gaussian / Laplace	D_t	✗	✗
Zhang et al. (2026)	**Inductive bias**: Gaussian / linear / cosine	N_t	✗	✗
Sedykh et al. (2026)	**Inductive bias**: linear	N_t	✗	✗

A.3 Related Works

In this section, we discuss related work along two tracks. Section A.3.1 covers methods that allocate non-uniform compute across diffusion processes, and Section A.3.2 covers scaling studies of ARMs, DLMs, and continuous-state diffusion models. Tables A3 and A4 summarize how Ermine differs from each track, respectively.

A.3.1 Compute Allocation in Diffusion Models

A body of work (Karras et al., 2022; Kingma & Gao, 2023; Esser et al., 2024; Hang et al., 2024; Arriola et al., 2025; Hong et al., 2026; Zhang et al., 2026; Sedykh et al., 2026) observes that not all diffusion timesteps contribute equally to the training objective, and reallocates compute along the time axis accordingly; Table A3 summarizes how these methods differ from Ermine. In continuous-state diffusion (Ho et al., 2020), the reallocation is realized by reshaping the log signal-to-noise ratio (SNR) distribution $p(\lambda_t)$, where $\lambda_t = \log \mathrm{SNR}_t$ (see Section A.6.4 **(T1)** for further discussion). Karras et al. (2022) first propose to sample the log-standard-deviation of the diffusion kernel ($\log \sigma$) from a Gaussian, Kingma & Gao (2023) interpret the noise schedule as an ELBO weighting and adopt Gaussian and cosine distributions for sampling noise levels, Esser et al. (2024) use a logit-normal timestep density, and Hang et al. (2024) conduct a detailed sweep over Gaussian, Laplace, Cauchy, linear, and cosine families for $p(\lambda_t)$. Discrete-state work (Austin et al., 2021) follows the same pattern: Arriola et al. (2025) restrict t to a truncated uniform interval to reduce gradient variance, and Hong et al. (2026) concentrate the timestep density on intermediate mask ratios. Zhang et al. (2026) and Sedykh et al. (2026) instead vary model capacity across timesteps, through expert-choice routing and per-step model scheduling respectively; the former targets training efficiency, while the latter targets inference efficiency. However, none of these works relate their non-uniform compute allocation scheme with scaling laws.

Three properties are common to all of these prior works, as summarized in Table A3. First, the schedule encodes an inductive bias: its functional form (e.g., Gaussian) is chosen in advance, and the associated hyperparameters (e.g., mean and standard deviation) are tuned by grid search. Second, the schedule allocates either the data axis or the parameter axis, but not both, sacrificing the flexibility of the remaining axis. Third, the design remains fixed as the training budget C changes, and therefore cannot account for the compute-dependent behavior observed in Fig. 3. Ermine differs from these approaches in all three aspects by casting time-dependent allocation as a scaling analysis. We fit a time-dependent scaling law and solve the resulting constrained allocation problem in closed form, so the schedule is *learned* from measured loss surfaces rather than assumed. N_t and D_t are allocated *jointly* under a single FLOP constraint, or separately through the time-invariant N and time-invariant D cases (Section A.1.2). In addition, the resulting allocation is an explicit function of the target budget C. In Section A.6.1, we compare Ermine with the Guassian and linear scheduling methods and show that an improper selection of hyper-parameters in these methods leads to inferior training efficiency.

Table A4: Comparison of scaling-law studies. Prior work spans three model families (ARMs, DLMs, and continuous-state diffusion) across text, image, and video modalities, and is fitted almost exclusively on dense backbones. All treat training compute as a single scalar aggregated over the generative process. Ermine is the only formulation that resolves the allocation along the diffusion time axis.

	Model Family	Modality	Architecture	Time-dependency
Ermine (ours)	DLM (MDM / UDM)	Text	Dense / MoE Transformer	✓
Kaplan et al. (2020)	ARM	Text	Dense Transformer	✗
Hoffmann et al. (2022)	ARM	Text	Dense Transformer	✗
Muennighoff et al. (2023)	ARM	Text	Dense Transformer	✗
Nie et al. (2025)	DLM (MDM)	Text	Dense Transformer	✗
Ni et al. (2025)	DLM (MDM)	Text	Dense Transformer	✗
Prabhudesai et al. (2025)	DLM (MDM)	Text	Dense Transformer	✗
Chao et al. (2026)	DLM (MDM / MDM-Prime)	Text	Dense Transformer	✗
Rütte et al. (2026)	DLM (MDM / UDM / GIDD)	Text	Dense Transformer	✗
Sahoo et al. (2026a)	DLM (MDM / UDM / Eso-LM)	Text	Dense Transformer	✗
Yin et al. (2025)	Continuous diffusion	Video	Dense Transformer	✗
Mei et al. (2025)	Continuous diffusion	Image	UNet	✗
Liang et al. (2026)	Continuous diffusion	Image	Dense Transformer	✗
Chickering et al. (2026)	Continuous diffusion	Text-to-image	Dense Transformer	✗

A.3.2 Scaling Laws of ARMs, DLMs, and Continuous Diffusion Models

Kaplan et al. (2020) establish power-law behavior of transformer-based LLMs (Vaswani et al., 2017) in each of N, D, and C individually, and conclude that model size should grow faster than data. Hoffmann et al. (2022) propose Chinchilla and revise this with the parametric form of Eq. (1), fitted by IsoFLOP profiles and a Huber-loss regression. Muennighoff et al. (2023) extend the Chinchilla formula to the data-constrained regime by replacing (N, D) with effective counts that decay exponentially with repetition. These laws were developed for ARMs, and six recent works (Nie et al., 2025; Ni et al., 2025; Prabhudesai et al., 2025; Chao et al., 2026; Rütte et al., 2026; Sahoo et al., 2026a) transfer them to DLMs. Nie et al. (2025) provide a scaling analysis of MDMs and conclude that MDMs require roughly $16\times$ more FLOPs than ARMs at the compute-optimal setup. Ni et al. (2025) refit the Chinchilla laws to MDMs and find matching exponents but different multipliers, concluding that a compute-optimal MDM requires two to five times more tokens and a correspondingly smaller model, with a higher irreducible loss attributable to the variational bound. Prabhudesai et al. (2025) isolate the data-constrained regime and report a much longer reuse half-life for MDMs than for ARMs, yielding a critical compute budget beyond which DLMs attain the lower loss. Chao et al. (2026) extend the MDM formulation with a partial masking scheme, and their scaling analysis shows stronger downstream performance than both MDMs and ARMs at matched compute. Rütte et al. (2026) and Sahoo et al. (2026a) study the scaling behavior of both MDMs and UDMs, offering a more comprehensive view of DLM scaling. The former fit scaling laws along the GIDD interpolation (von Rütte et al., 2025) between the UDM and MDM objectives, and obtain Chinchilla coefficients that differ from those of Nie et al. (2025). The latter additionally compare against Eso-LM (Sahoo et al., 2026b), which interpolates between ARMs and MDMs. Both works conclude that MDMs are the most compute-efficient family among DLMs. All of these laws, however, apply Chinchilla directly and treat training as a single monolithic budget, i.e., the loss is a function of the scalars (N, D), which implicitly assumes that every diffusion timestep is equally worth scaling. Ermine relaxes this assumption by resolving the scaling law along the time axis.

An orthogonal line of studies (Liang et al., 2026; Mei et al., 2025; Yin et al., 2025; Chickering et al., 2026) investigates the scaling behaviors of continuous-state diffusion models. Liang et al. (2026) conduct the first iso-FLOP study of diffusion transformers, fitting power laws over budgets from 10^{17} to 6×10^{18} FLOPs and showing that generation quality (e.g., FID) positively correlates the pre-training loss. Mei et al. (2025) instead examine the inference side, finding that under a fixed sampling budget smaller latent diffusion models frequently outperform larger ones. Yin et al. (2025) extend the analysis to video diffusion transformers, and Chickering et al. (2026) scale the study to 10^{22} FLOPs, reporting that text-to-image diffusion models reach compute optimality at roughly 200 tokens per parameter, an order of magnitude above the Chinchilla prescription for ARMs. These

Table A5: Time-wise loss predicted by the independently fitted coefficients versus that measured from composed models.

	$t=0.1$	$t=0.2$	$t=0.3$	$t=0.4$	$t=0.5$	$t=0.6$	$t=0.7$	$t=0.8$	$t=0.9$
Predicted	1.4930	1.7027	2.0259	2.3967	2.8963	3.4823	4.1801	5.0622	6.1106
Empirical	1.4152	1.6511	2.0047	2.3202	2.8303	3.4460	4.1787	4.9965	6.0996
Deviation	5.21%	3.03%	1.05%	3.19%	2.28%	1.04%	0.03%	1.30%	0.18%

studies characterize scaling in the discrete-state setting's continuous counterpart, but all treat compute as a single scalar aggregated over the diffusion process. In contrast, Ermine resolves the allocation along the time axis.

A.4 Technical Specification

This section provides technical details of Ermine. Section A.4.1 presents the coefficient fitting procedure. Section A.4.2 elaborates on the $C = 6ND$ approximation. Section A.4.3 describes how the bisection method is used to derive Ermine's optimal allocation. Finally, Section A.4.4 introduces three GPU assignment schemes that differ in how aggressively they optimize compute utilization.

A.4.1 Fitting and Predicting from Empirical Scaling Laws

Section 2.1 and Section 3.1 show how Chinchilla and Ermine are parameterized. This section details the parameter fitting procedure.

The Chinchilla scaling law models the loss as $\hat{\mathcal{L}}(N, D) = E + AN^{-a} + BD^{-b}$, where (E, A, B, a, b) are parameters to be fitted. To estimate these parameters, one conducts a suite of M training runs $\{(N^i, D^i, \mathcal{L}^i)\}_{i=1}^M$ and minimizes the Huber loss (Huber, 1964):

$$\min_{E,A,B,a,b} \sum_{i=1}^{M} \text{Huber}_\varepsilon\left(\log \hat{\mathcal{L}}(N^i, D^i) - \log \mathcal{L}^i\right), \quad \text{Huber}_\varepsilon(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \le \varepsilon, \\ \varepsilon\left(|x| - \frac{1}{2}\varepsilon\right) & \text{otherwise,} \end{cases} \tag{A11}$$

where ε is set to 10^{-3}. Analogously, the Ermine time-wise loss is written as $\hat{\ell}_t(N_t, D_t) = E_t + A_t N_t^{-a_t} + B_t D_t^{-b_t}$, where $(E_t, A_t, B_t, a_t, b_t)$ are the parameters. To derive these quantities, one prepares a suite of M training runs $\{(N_t^i, D_t^i, \ell_t^i)\}_{i=1}^M$ for $t \in [0, 1]$ and minimizes the Huber loss at each timestep:

$$\min_{E_t,A_t,B_t,a_t,b_t} \sum_{i=1}^{M} \text{Huber}_\varepsilon\left(\log \hat{\ell}_t(N_t^i, D_t^i) - \log \ell_t^i\right), \quad \forall t \in [0, 1]. \tag{A12}$$

An exact derivation of Ermine would require enumerating all combinations of (N_t, D_t) along the time axis and selecting the allocation that minimizes the total loss $\hat{\mathcal{L}}$, which would demand a far larger M than Chinchilla. Empirically, however, we find that the time-wise losses are decoupled across timesteps, which allows M to remain the same as in Chinchilla. Specifically, the loss $\hat{\ell}_t$ attained at one timestep is unaffected by the capacity and data allocated to the others, so each $\hat{\ell}_t$ can be fitted independently. To verify this, we compose models of different sizes along the time axis and compare the resulting time-wise losses against those predicted by the independently fitted coefficients. As reported in Table A5, the deviation across $t \in \{0.1, \ldots, 0.9\}$ averages 1.92% and never exceeds 5.21%, with the maximum occurring at the smallest t, confirming that joint allocations need not be enumerated.

After the fitting procedure, the prediction of Ermine follows the two key steps described as follows:

- **(Step 1)** Derive μ^* given C via $\int_0^1 (\gamma_t F_t/\mu)^{\frac{1}{1+\gamma_t}}\, dt = C$. This can be achieved by running the bisection method (Algorithm 1) by setting $f(\mu) = (\int_0^1 (\gamma_t F_t/\mu)^{\frac{1}{1+\gamma_t}}\, dt) - C$ with timestep discretization.
- **(Step 2)** Calculate C_t^*, N_t^*, and D_t^* using μ^* according to Eq. (10).

Algorithm 1 Bisection Method (see Section A.4.3)

Input: $f : [w, v] \to \mathbb{R}$, $f(w)f(v) < 0$, tolerance ε

while $v - w > \varepsilon$ **do**
  $m \leftarrow (w + v)/2$
  if $f(w)f(m) \leq 0$ **then**
    $v \leftarrow m$
  else
    $w \leftarrow m$
  end if
end while
return $(w + v)/2$

For the Ermine time-invariant cases, the inference procedure is similar, with the function f in **Step 1** defined differently. For the constant N case, we set $f(N) = 6N\left(b_t B_t/\tilde{\mu}(N)\right)^{\frac{1}{1+b_t}} - C$. For the constant D case, we set $f(D) = 6D\int_0^1 \left(a_t A_t/\tilde{\mu}(D)\right)^{\frac{1}{1+a_t}}\, dt - C$. Then, D_t^* or N_t^* can be derived from N^* or D^*, respectively. In practice, although measuring and fitting the coefficients at every timestep $t \in [0, 1]$ is feasible given the pretrained weights, evaluating the integral in f in closed form for all of them is not. We therefore approximate it with a Riemann sum and fit the coefficients on the discretized timesteps.

A.4.2 Estimating the Compute Consumption

Compute Approximation using $6ND$. Following the convention of scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), we approximate the training cost of a model with N non-embedding parameters as $6N$ floating-point operations (FLOPs) per token. This calculation contains $2N$ for the forward pass and $4N$ for the backward pass. As a result, processing D tokens costs $C = D(2N + 4N) = 6ND$. For the MoE model, we follow Muennighoff et al. (2025) and approximate the compute as $6N_{\text{act}}D$, where N_{act} denotes the number of activated parameters.

Compute Density Approximation via $6N_tD_t$. DLM training approximates the loss with a single-sample Monte Carlo estimate, i.e., for each clean sequence $\boldsymbol{x}_0$ in a batch, we draw one noisy counterpart $\boldsymbol{x}_t$ and form the pair $(\boldsymbol{x}_0, \boldsymbol{x}_t)$. Only $\boldsymbol{x}_t$ traverses the architecture, and it incurs the same forward-backward cost regardless of t. Let D_t denote the number of tokens in $\boldsymbol{x}_0$; since $\boldsymbol{x}_t$ retains the same length, the model processes exactly D_t noisy tokens at timestep t. Given the parameter count N_t invoked at t, the compute expended at t is approximated by $C_t = 6N_tD_t$.

Total Cost Consumption as $C = \int_0^1 6N_tD_t\, dt$. One way to understand this formulation is through expected computation. Let $g : [0, 1] \to \mathbb{R}$ be a continuous function of t, written g_t for brevity, and let $\mathbb{E}_{t\sim U(0,1)}[g_t]$ denote its expectation under a uniform time distribution $t \sim U(0, 1)$. Since the density of $U(0, 1)$ is 1 on $[0, 1]$, this expectation reduces to an integral,

$$\mathbb{E}_{t\sim U(0,1)}[g_t] = \int_0^1 U(0,1)\, g_t\, dt = \int_0^1 1 \cdot g_t\, dt = \int_0^1 g_t\, dt. \tag{A13}$$

Setting $g_t = C_t$, the expected per-timestep compute is therefore $\mathbb{E}_{t\sim U(0,1)}[C_t] = \int_0^1 C_t\, dt$, and we write the budget constraint as $C = \mathbb{E}_{t\sim U(0,1)}[C_t] = \int_0^1 C_t\, dt = \int_0^1 6N_tD_t\, dt$. This definition is consistent with the standard Chinchilla calculation as a special case. When N and D are time-independent, so that $N_t = N$ and $D_t = D$ for all t, we recover

$$C = \int_0^1 6N_tD_t\, dt = \int_0^1 6ND\, dt = 6ND\int_0^1 dt = 6ND \cdot 1 = 6ND. \tag{A14}$$

A.4.3 Solving Compute, Parameter, and Data Constraints

The variables μ in Eq. (10) and D and N in Eq. (A5) admit solutions handled by a numerical solver. This section presents a simple realization of such a solver based on the bisection method (Kaw et al., 2009). The idea is to express the constraint as a scalar function $f : \mathbb{R} \to \mathbb{R}$ whose root y, i.e., the value satisfying $f(y) = 0$, corresponds to the desired quantity. Algorithm 1 summarizes this

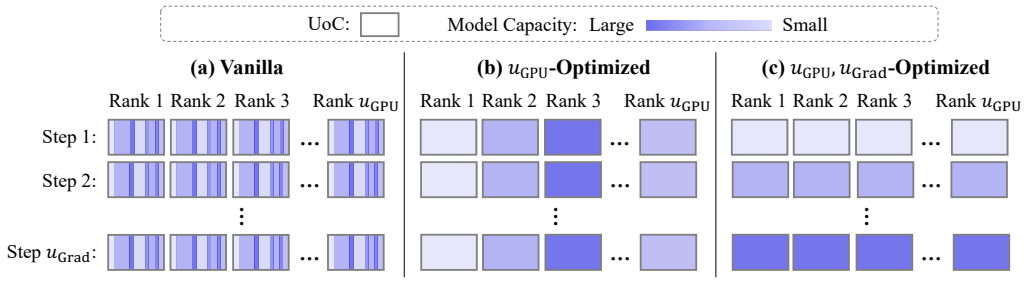

Figure A1: Assigning time-dependent model capacity across units of compute (UoC), where $u = u_{\text{GPU}} \times u_{\text{Grad}}$ and color denotes the model capacity. **(a) Vanilla:** each UoC mixes intervals of different capacity. **(b) u_{GPU}-optimized:** capacity is homogeneous within a rank but differs across ranks. **(c) $u_{\text{GPU}}, u_{\text{Grad}}$-optimized:** capacity is constant across all ranks within an accumulation step and varies only across steps.

procedure and **Proposition** A3 guarantees its convergence. The same guarantee applies to D and N in Eq. (A5).

Proposition A3. *Let $f : \mathbb{R} \to \mathbb{R}$ be continuous, and let $w_0 < v_0$ be two initial points at which f takes opposite signs, i.e., $f(w_0)f(v_0) < 0$. Let (w_i, v_i) denote the lower and upper endpoints of the search interval maintained by Algorithm 1 after i iterations. Then, the intervals $\{(w_i, v_i)\}_{i=1}^{M}$ are nested, satisfy $v_i - w_i = (v_0 - w_0)/2^i$, and each contains a root y^* of f. Consequently, the midpoint $m_M = (w_M + v_M)/2$ returned after M iterations satisfies $|m_M - y^*| \le (v_0 - w_0)/2^{M+1}$.*

Proof. We show by induction that $f(w_i)f(v_i) < 0$ for all i. The case $i = 0$ is the assumption. Given $f(w_{i-1})f(v_{i-1}) < 0$ and the midpoint $m_i = (w_{i-1} + v_{i-1})/2$, the algorithm keeps $(w_i, v_i) = (u_{i-1}, m_i)$ when $f(w_{i-1})f(m_i) < 0$, which preserves the property; otherwise $f(m_i)$ shares the sign of $f(w_{i-1})$, so $f(m_i)f(v_{i-1}) < 0$ and the choice $(w_i, v_i) = (m_i, v_{i-1})$ preserves it as well.

Since f is continuous and changes sign on (w_i, v_i), the intermediate value theorem guarantees a root $y^* \in (w_i, v_i)$ for every i. Both branches replace one endpoint by the midpoint, so $(w_i, v_i) \subseteq (w_{i-1}, v_{i-1})$ and $v_i - w_i = \frac{1}{2}(v_{i-1} - w_{i-1})$; unrolling gives $v_i - w_i = (v_0 - w_0)/2^i$. As y^* and m_M both lie in (w_M, v_M), their distance is at most half the interval length, i.e., $(v_0 - w_0)/2^{M+1}$. □

Convergence of Ermine. Define $f(\mu) = \int_0^1 (\gamma_t F_t/\mu)^{\frac{1}{1+\gamma_t}}\, dt - C$. Since the derivative of the integrand is negative:

$$\frac{\partial}{\partial \mu}\left(\frac{\gamma_t F_t}{\mu}\right)^{\frac{1}{1+\gamma_t}} = \frac{-1}{1+\gamma_t}\,(\gamma_t F_t)^{\frac{1}{1+\gamma_t}}\,\mu^{\frac{-1}{1+\gamma_t}-1} < 0, \tag{A15}$$

the integrand is continuous and strictly decreasing in μ for every t, and so is f. Moreover, $\lim_{\mu\to 0^+} f(\mu) = +\infty$ and $\lim_{\mu\to\infty} f(\mu) = -C < 0$. As a result, for any $C > 0$, f is continuous and changes sign on $(0, \infty)$. It admits a unique root μ^*, and **Proposition** A3 guarantees that Algorithm 1 converges to it. In practice, we solve for $\log \mu$ rather than μ itself, which keeps the search numerically stable across the many orders of magnitude spanned by μ. The bracket is initialized at $(\texttt{log_low_mu},\ \texttt{log_high_mu}) = (-700.0,\ 700.0)$, selected to span the entire range representable in double precision, and then narrowed inward until the sign condition $f(\texttt{log_low_mu})\, f(\texttt{log_high_mu}) < 0$ is satisfied numerically.

A.4.4 Maximizing GPU Utilization

Time-dependent compute allocation may incur burdens to hardware if the GPU allocation is not handled properly. Since intervals with larger N_t^* require more compute per mini-batch, assigning them arbitrarily across devices leaves the remaining hardware idle while it waits at the synchronization barrier. Moreover, current machine learning frameworks are optimized for a fixed model architecture and expect a consistent model size on every device (GPU rank); deviating from this typically requires substantial customization and degrades training throughput. To maximize training

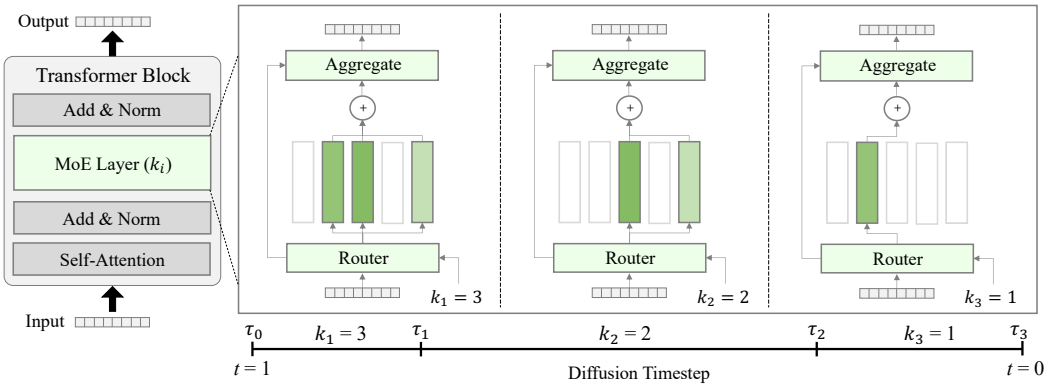

Figure A2: Architecture of the time-dependent MoE Transformer. Left: the Transformer block. Right: the MoE layer, where the number of activated experts is set by k_i. This example uses $u = 3$, with $k_1 = 3$, $k_2 = 2$, and $k_3 = 1$. τ_i represents the time boundary for different top-k values.

efficiency without massive code refactoring, this section describes how to assign model capacity and data to each unit of compute (UoC).

In this work, a UoC (u) denotes an available training resource under data parallelism, each of which consumes the same amount of data (i.e., a mini-batch). Its value is the number of GPUs (u_{GPU}) times the gradient accumulation steps (u_{Grad}), i.e., $u = u_{\text{GPU}} \times u_{\text{Grad}}$. Maximizing GPU utilization while maintaining implementation simplicity amounts to ensuring that all u_{GPU} devices hold the same model size at any given moment, i.e., that the top-k routing value is constant across ranks within a step.

Fig. A1 compares three assignment schemes. The **vanilla** scheme (Fig. A1 (a)) samples timesteps independently within each UoC. As a result, a single mini-batch mixes intervals of different N_t^* and every rank hosts a different capacity, incurring both load imbalance and architectural heterogeneity. Ordering the intervals along the rank axis (u_{GPU}**-optimized**, Fig. A1 (b)) makes each rank internally homogeneous. However, since the ranks still differ from one another, the step time is dictated by the slowest (largest-capacity) rank. Finally, the $u_{\text{GPU}}, u_{\text{Grad}}$**-optimized** scheme (Fig. A1 (c)) instead partitions the intervals along the gradient accumulation axis. Under this scheme, all u_{GPU} ranks share an identical capacity within each accumulation step, and the capacity varies only across steps. All devices therefore execute the same architecture in lockstep, eliminating idle time at the synchronization barrier while still realizing the non-uniform N_t^* profile over the full update.

A.5 Experimental Configurations

This section provides supplemental experimental details and hyperparameter configurations. The settings for the experiments described in Sections 4.1, 4.2, and 4.3 are detailed in Sections A.5.1, A.5.2, and A.5.3, respectively. The code is available in this anonymous link: `https://anonymous.4open.science/r/ermine/README.md`.

A.5.1 Experiments in Section 4.1

Dataset. The experiments in Section 4.1 use the C4 dataset (Raffel et al., 2020) (the English subset), tokenized with the GPT-2 tokenizer (Radford et al., 2019). The maximum sequence length and the vocabulary size are set to $L = 2{,}048$ and $V = 50{,}257$, respectively. Documents are concatenated with the `<eod>` token as a separator and packed into samples of L tokens. Since the training loss does not correspond to the ELBO under quantile-based timestep sampling, all reported losses and perplexities (e.g., Table 2) are computed with the standard evaluation procedure and therefore do reflect the ELBO.

Implementation Details. The experiments adopt a LLaMA-style (Touvron et al., 2023) Transformer (Vaswani et al., 2017) with rotary positional embeddings (RoPE) (Su et al., 2024), SwiGLU activations (Shazeer, 2020), RMSNorm (Zhang & Sennrich, 2019), and QK-normalization (Dehghani et al., 2023). Model capacities N range from 5M to 1.6B parameters; the corresponding depths, hidden dimensions, and numbers of attention heads are listed in Table A6. The dense model

Table A6: Model architecture configurations. N represents the number of (activate) non-embedding parameters, H denotes the hidden dimension size, R is the number of Transformer blocks, and 'Heads' refer to the number of attention heads. All models use 8 experts with 2 activated per token, so that $N_{\text{Total}} \approx 3N_{\text{act}}$.

N (N_{act})	N_{Total}	H	R	Heads	N (N_{act})	N_{Total}	H	R	Heads
5M	15M	256	6	8	153M	459M	896	16	16
7M	21M	320	6	10	199M	597M	1,024	16	16
10M	30M	320	8	10	249M	747M	1,024	20	16
14M	42M	448	6	7	315M	945M	1,280	16	10
25M	75M	512	8	8	352M	1,056M	1,280	18	10
36M	108M	576	9	9	411M	1,233M	1,280	21	10
48M	144M	640	10	10	566M	1,698M	1,408	24	11
63M	189M	640	13	10	771M	2,313M	1,792	20	14
77M	231M	640	16	10	1,107M	3,321M	2,048	22	16
106M	318M	768	15	12	1,529M	4,587M	2,304	24	18

uses an MLP ratio of $8/3$, while the baseline MoE model uses 8 experts in total, 2 of which are activated, each with an MLP ratio of $8/6$; the activated capacity therefore matches that of the dense models. The models are trained with two auxiliary losses: load balance loss (Shazeer et al., 2017) and router z-loss (Zoph et al., 2022), with balance factors set to $1e-3$ and $1e-2$, respectively. Following Ludziejewski et al. (2025), the FLOPs consumed by an MoE model are estimated as $6N_{\text{act}}D$, where N_{act} denotes the number of activated (non-embedding) parameters. All diffusion processes (MDMs and UDMs) use the linear schedule $\alpha_t = 1 - t$. The models are optimized with AdamW (Loshchilov & Hutter, 2019) ($\beta_1 = 0.9$, $\beta_2 = 0.999$) at a learning rate of 2×10^{-4}, cosine-decayed to a minimum of 2×10^{-5}, with a batch size of 512. Our implementation builds on Megatron-LM (Shoeybi et al., 2019) and the scaling experiments of MDM-Prime-v2 (Chao et al., 2026). Models with $N \leq 400$M are trained on eight NVIDIA L40 GPUs (48 GB), and larger models on eight H100 (80 GB) or H200 (141 GB) GPUs.

A.5.2 Experiments in Section 4.2

Dataset. Training is conducted on the OpenWebText (OWT) dataset (Gokaslan et al., 2019), with text tokenized using the GPT-2 tokenizer (Radford et al., 2019). The maximum sequence length is set to $L = 1{,}024$ and the vocabulary size to $V = 50{,}257$. Each document is prepended and appended with the `<eos>` token to mark its boundaries; the documents are then concatenated and the resulting token stream is partitioned into segments of length L. As OWT provides no official validation set, we follow Sahoo et al. (2024) and hold out the final 100,000 samples for validation. Our preprocessing and evaluation pipeline is otherwise identical to that of Sahoo et al. (2024), ensuring a fair comparison. All reported losses and perplexities (e.g., Fig. 7) are computed with the standard evaluation procedure.

Implementation Details. For a fair comparison, we adopt the same Diffusion Transformer (DiT) (Peebles & Xie, 2023) architecture as prior work (Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024; Arriola et al., 2025); the detailed configuration is provided in Table A7. Since the estimated integrated N and D of Ermine are nearly identical to those of Chinchilla (see Table 2), we implement both MDM-MoE (Chinchilla) and MDM-MoE (Ermine) with the same configuration (i.e., the N_{act} entry in Table A7), differing only in their top-k scheduling. Both models are trained with the same total compute budget of 2.89×10^{20} FLOPs. The code implementation follows the official implementations of (Sahoo et al., 2024; 2025; Chao et al., 2025). The MoE models are implemented with MegaBlocks (Gale et al., 2023) and the baseline model uses 8 experts in total, 2 of which are activated per token, each with half the MLP ratio of the dense counterpart. The activated capacity therefore matches that of the dense models, following the same convention as in Section A.5.1. We apply a dropout rate of 0.1 throughout training and use the linear

Table A7: DiT model architecture configurations. N represents the number of non-embedding parameters, H denotes the hidden dimension size, R is the number of Transformer blocks.

N (N_{act})	H	R	Heads
92M	768	12	12
466M	1152	34	9
546M	1408	22	11
1,251M	2048	24	16

Table A8: An overview of the zero-shot benchmarks used for evaluation, covering both the multiple-choice commonsense reasoning tasks and the held-out corpora used for perplexity evaluation.

Benchmark	Description
Zero-shot Q&A Evaluation	
SciQ (Welbl et al., 2017)	Crowdsourced multiple-choice science exam questions, each paired with a supporting evidence passage.
SIQA (Sap et al., 2019)	Social commonsense questions about the motivations, reactions, and likely next actions of people in everyday situations.
ARC-e (Clark et al., 2018)	The easy subset of the AI2 Reasoning Challenge, drawn from grade-school science exams.
PIQA (Bisk et al., 2020)	Physical commonsense questions asking which of two candidate solutions better achieves an everyday goal.
HellaS (Zellers et al., 2019)	Sentence completion with adversarially filtered endings that are fluent but implausible.
Zero-shot PPL Evaluation	
LAMBADA (Paperno et al., 2016)	Narrative passages from novels whose final word is recoverable from the full passage but not the last sentence, testing long-range context.
WikiText (Merity et al., 2016)	Long-form Wikipedia articles from the verified Good and Featured subsets, with original casing, punctuation, and numerals retained.
PTB (Marcus et al., 1993)	Wall Street Journal newswire under standard preprocessing: lowercased, punctuation stripped, numerals and rare words replaced.
LM1B (Chelba et al., 2013)	A large news-crawl corpus of shuffled sentences, each modeled in isolation without cross-sentence context.
AG News (Zhang et al., 2015)	Short news items pairing a headline with a brief description, spanning world, sports, business, and sci/tech topics.
Pile (ArXiv) (Cohan et al., 2018)	Full texts of arXiv scientific papers, marked by technical vocabulary and mathematical notation.
Pile (PubMed) (Cohan et al., 2018)	Full texts of biomedical articles from PubMed Central, dominated by clinical terminology and formal structure.

schedule $\alpha_t = 1 - t$ for the diffusion process. The models are optimized with AdamW (Loshchilov & Hutter, 2019) using a weight decay of 0.1, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The learning rate is linearly warmed up from 0 to 3×10^{-4}. Models with 92M non-embedding parameters are trained on eight NVIDIA L40 GPUs (48 GB), and larger models on eight H100 (80 GB) or H200 (141 GB) GPUs.

A.5.3 Experiments in Section 4.3

Dataset. The models are trained using the Dolma 1.7 dataset (Soldaini et al., 2024), with text tokenized using the OLMo (Muennighoff et al., 2025) tokenizer. The maximum sequence length L is set to 2,048, and the vocabulary size V is 50,280. Following the approach in (Nie et al., 2025), we randomize the sequence length for 1% of the training data. This is implemented by sampling a length from a uniform distribution $\mathcal{U}(1, 2048)$ for each sequence. We measure zero-shot accuracy on five commonsense reasoning benchmarks: SciQ (Welbl et al., 2017), SIQA (Sap et al., 2019), Hellaswag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and ARC-Easy (ARC-e) (Clark et al., 2018). In Table 3, both MDM-MoE (Chinchilla; A1.1B) and MDM-MoE (Ermine; A1.1B) use 1,024 Monte Carlo (MC) steps for evaluation following (Nie et al., 2025). Detailed task descriptions are provided in Table A8. For prediction, we concatenate each answer candidate with the question prompt and select the option that yields the highest likelihood under the diffusion model.

Implementation Details. The model architecture is adapted from the OLMoE architecture (Muennighoff et al., 2025), which uses RoPE, SwiGLU, and RMSNorm. The model has 1.04B activated and 3.09B total non-embedding parameters, and is constructed from 18 Transformer blocks with a hidden size of 2,240 and 35 attention heads, using dropless MoE layers. The baseline MoE layers

Table A9: Ablation studies at a matched budget $C = 10^{19}$ FLOPs. **(AS1)** The MoE backbone improves over the dense Transformer for both MDMs and UDMs. **(AS2)** Ermine improves over Chinchilla for both MDMs and UDMs. **(AS3)** The N_t^* axis in Ermine can be realized by different architectures, where MoE performs the best. **(AS4)** Under constant N, the Ermine D_t^* outperforms hand-designed Gaussian timestep densities proposed in prior works. **(AS5)** Under invariant D, the Ermine N_t^* allocation outperforms hand-designed linear capacity schedules proposed in prior works.

Model	Allocation	Validation Loss (↓)
***(AS1)** Architecture Comparison*		
MDM-MoE	Chinchilla	**3.675±0.016**
MDM	Chinchilla	3.789±0.018
UDM-MoE	Chinchilla	3.826±0.006
UDM	Chinchilla	3.932±0.013
***(AS2)** Scaling Law Comparison*		
MDM-MoE	Ermine	**3.599±0.004**
MDM-MoE	Chinchilla	3.675±0.016
UDM-MoE	Ermine	3.774±0.013
UDM-MoE	Chinchilla	3.826±0.006
***(AS3)** Realization of N_t^**		
MDM-MoE	Ermine	**3.599±0.004**
MDM-MoE	Chinchilla	3.675±0.016
MDM-Looped (Dense)	Ermine	3.718±0.006
MDM-Looped (Sparse)	Ermine	3.671±0.007
***(AS4)** Time-Invariant N Allocation Variants*		
MDM-MoE	Ermine (Time-Invariant N)	**3.608±0.008**
MDM-MoE	Chinchilla	3.675±0.016
MDM-MoE	Chinchilla + Gaussian D_t ($\mu = 0.5$, $\sigma = 0.1$) (Hong et al., 2026)	3.626±0.010
MDM-MoE	Chinchilla + Gaussian D_t ($\mu = 0.1$, $\sigma = 0.5$) (Hong et al., 2026)	3.707±0.025
MDM-MoE	Chinchilla + Gaussian D_t ($\mu = 0.3$, $\sigma = 0.3$) (Hong et al., 2026)	3.709±0.023
***(AS5)** Time-Invariant D Allocation Variants*		
MDM-MoE	Ermine (Time-Invariant D)	**3.611±0.003**
MDM-MoE	Chinchilla	3.675±0.016
MDM-MoE	Chinchilla + Linear N_t ($k_{\text{min}} = 2$, $k_{\text{max}} = 8$) (Zhang et al., 2026)	3.740±0.016
MDM-MoE	Chinchilla + Reverse Linear N_t ($k_{\text{min}} = 2$, $k_{\text{max}} = 8$) (Zhang et al., 2026)	3.732±0.015
MDM-MoE	Chinchilla + Reverse Linear N_t ($k_{\text{min}} = 1$, $k_{\text{max}} = 4$) (Zhang et al., 2026)	3.753±0.027

use 8 experts in total, 2 of which are activated per token, following the same convention as in Section A.5.1. For the diffusion process, we use the linear schedule $\alpha_t = 1 - t$. Training is performed with AdamW (Loshchilov & Hutter, 2019) using $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The maximum learning rate is set to 2×10^{-4} and follows a cosine decay to a minimum of 2×10^{-5}. The model is trained on NVIDIA H100 (80 GB) and H200 (141 GB) GPUs.

A.6 Supplementary Experiments

This section provides 14 sets of additional experiments: five ablation studies (**AS1-AS5**, Section A.6.1), three wall-time comparisons (**TC1-TC3**, Section A.6.2), three supplementary experiments (**S1-S3**, Section A.6.3), and three consistent trends derived from our scaling law (**T1-T3**, Section A.6.4).

A.6.1 Ablation Study

We conduct five ablation studies (**AS1-AS5**) at a matched compute budget of $C = 10^{19}$ FLOPs, reported in Table A9. Each block isolates a single design choice. All models are trained on C4; we report the loss on the held-out validation split averaged over three runs, where lower is better.

(AS1) MoE Transformers Outperform Dense Transformers. The first block isolates the backbone while holding the allocation fixed at Chinchilla. Replacing the dense Transformer with an

MoE backbone lowers the validation loss from 3.789 to 3.675 for MDMs and from 3.932 to 3.826 for UDMs. The gain is consistent across both families and exceeds the standard deviation by roughly an order of magnitude, indicating that the MoE architecture benefits DLMs independently of the corruption process. The ordering between families is unaffected by the backbone: MDMs outperform UDMs in both the dense and the MoE setting. We therefore adopt MDM-MoE as the reference configuration in the remaining ablations.

(AS2) Ermine Outperforms Chinchilla across MDMs and UDMs. The first block compares the Ermine allocation against the Chinchilla allocation (Hoffmann et al., 2022) for MDMs and UDMs. Ermine lowers the validation loss from 3.675 to 3.599 for MDMs and from 3.826 to 3.774 for UDMs. The improvement holds across both families and exceeds the standard deviation by an order of magnitude, indicating that the gain comes from the time-dependent allocation rather than run-to-run variation.

(AS3) MoE Outperforms Looped Alternatives for Time-Dependent Architecture. Allocating model capacity as a function of the diffusion timestep requires a variable-sized model architecture. Besides the width-wise design (MoE), we consider two depth-wise alternatives: a looped Transformer (Giannou et al., 2023) that varies effective depth with weights shared across iterations, and a sparse variant that introduces sparsity along the depth dimension. The sparse design outperforms the dense one (3.671 versus 3.718), while the MoE realization remains the strongest of the three at 3.599, and we adopt it in our main experiments.

(AS4) Ermine (Time-Invariant N) Outperforms Gaussian-D_t Allocation (Hong et al., 2026). Prior work (see Section A.3) concentrates the data distribution over timesteps using a fixed density family, typically a Gaussian whose mean and variance are selected by grid search (Hong et al., 2026). We compare Ermine (Inv. N) against Chinchilla combined with three such densities. The results span 0.083 depending only on the choice of (μ, σ): the best setting reaches 3.626, while the remaining two (3.707 and 3.709) fall short of the plain Chinchilla reference at 3.675. Without intensive tuning, this approach can degrade performance. Ermine attains 3.608 with no hyperparameters, improving on even the best tuned Gaussian while removing the search entirely.

(AS5) Ermine (Time-Invariant D) Outperforms Linear-N_t Allocation (Zhang et al., 2026). We next hold D time-invariant and ablate the capacity schedule. Prior designs vary the number of activated experts linearly over the diffusion process, parameterized by the endpoints $k_{\min}$ and $k_{\max}$ (Zhang et al., 2026). None of the three variants improves on the time-invariant capacity Chinchilla reference: the linear schedule with $(k_{\min}, k_{\max}) = (2, 8)$ reaches 3.740, and the reverse-linear schedules reach 3.732 and 3.753. In contrast, Ermine (Inv. D) reaches 3.611 under the same budget, improving on the best hand-designed schedule by 0.121. To conclude, AS4 and AS5 suggest that deriving the allocation from Ermine is preferable to prescribing it through a hand-designed family.

A.6.2 Training and Inference Efficiency Comparison

Table A10 presents wall-clock training and inference time comparisons (**TC1-TC3**) under a matched setting, substantiating the practical benefits of using MDMs with time-dependent allocation. All runs adopt the same $N_{\text{act}} = 106$M MoE backbone. The Chinchilla model uses $k = 2$, while the k value of Ermine is scheduled according to the predicted N_t^* (average $k = 2$). Times are reported as means over 50 repeats and are measured on eight H200 GPUs with an identical batch size of 32.

(TC1) GPU Assignment Scheme Dictates the Runtime. A direct implementation of time-dependent allocation, which samples timesteps independently across the batch and dispatches each to its predicted N_t^*, is extremely slow; the schemes described in Section A.4.4 are what make the allocation practical. Mixing intervals of different N_t^* within every mini-batch (Fig. A1) forces each rank to run a distinct architecture, and the resulting overhead increases the step time by more than $10\times$ relative to Chinchilla's static allocation (13,058.1 ms versus 1,228.6 ms). Both proposed schemes (u_{GPU}-optimized and $u_{\text{GPU}}, u_{\text{Grad}}$-optimized) recover nearly all of this overhead. Ordering intervals along the rank axis brings the step time to 1,238.5 ms, and additionally partitioning along the gradient accumulation axis reaches 1,236.6 ms, within 1% of the Chinchilla baseline and inside run-to-run variation. As a result, the loss gains reported in Section 4 are obtained at comparable wall-clock training time.

(TC2) UDMs are Less Efficient than MDMs in Wall Time. UDM-MoE is slower per step than MDM-MoE (1,311.8 ms versus 1,228.6 ms) and consumes more memory. The gap stems from

Table A10: Training and inference efficiency comparison at matched activated capacity ($N_{\text{act}} = 106\text{M}$), measured on eight H200 GPUs with batch size 32; times are means over 50 repeats. **(TC1)** With the proposed GPU assignment schemes, the Ermine schedule matches static allocation to within 1%, whereas the vanilla assignment is over $10\times$ slower. **(TC2)** The UDM loss calculation is more expensive per step than the masked one at matched capacity. **(TC3)** Time-dependent allocation leaves sampling latency unchanged.

Model	Allocation	Wall-clock Time (ms)
***(TC1)** Training Time; Chinchilla and Ermine*		
MDM-MoE	Chinchilla	1228.6 ± 129.9
MDM-MoE	Ermine (Vanilla)	13058.1 ± 424.0
MDM-MoE	Ermine (u_{GPU}-Optimized)	1238.5 ± 160.3
MDM-MoE	Ermine (u_{GPU}, u_{Grad}-Optimized)	1236.6 ± 155.4
***(TC2)** Training Time; MDMs and UDMs*		
MDM-MoE	Chinchilla	1228.6 ± 129.9
UDM-MoE	Chinchilla	1311.8 ± 195.4
***(TC3)** Inference Time; Chinchilla and Ermine*		
MDM-MoE	Chinchilla	182.43 ± 2.83
MDM-MoE	Ermine	186.19 ± 3.26

Table A11: Zero-shot validation perplexities evaluated on seven textual datasets. Lower values correspond to better performance. All models are trained on OWT with their training perplexity shown in Table 2. Models are grouped into order-specific and order-agnostic families, and **bold** marks the best value per dataset within each group.

	LBD (↓)	Wiki (↓)	PTB (↓)	LM1B (↓)	AGN (↓)	ArXiv (↓)	PMed (↓)
Order-Specific Models							
ARM (Sahoo et al., 2024)	51.28	25.75	**82.05**	51.25	52.09	41.73	49.01
BD3-LM (Arriola et al., 2025)	≤50.03	≤31.31	≤96.81	≤60.88	≤61.67	≤39.20	≤42.52
ARM* (Chinchilla)	**35.63**	**21.27**	100.95	**42.96**	**38.53**	**30.74**	**30.88**
Order-Agnostic Models							
UDM (Sahoo et al., 2025)	≤49.78	≤33.57	≤89.35	≤73.86	≤67.81	≤40.39	≤44.48
SEDD (Lou et al., 2024)	≤49.86	≤34.28	≤100.09	≤68.20	≤62.09	≤38.48	≤45.39
MDM (Sahoo et al., 2024)	≤47.52	≤32.83	≤95.26	≤67.01	≤61.15	≤ 37.37	≤41.89
MDM* (Chinchilla)	≤42.99	≤27.43	≤81.72	≤60.66	≤49.24	≤**31.07**	≤**34.69**
MDM-MoE* (Chinchilla)	≤42.87	≤27.19	≤84.92	≤59.70	≤46.85	≤33.04	≤36.66
MDM-MoE* (Ermine; Inv. N)	≤42.76	≤**25.83**	≤76.58	≤55.30	≤43.68	≤33.74	≤36.49
MDM-MoE* (Ermine; Inv. D)	≤**42.02**	≤28.32	≤75.10	≤**53.70**	≤43.59	≤32.60	≤35.33
MDM-MoE* (Ermine)	≤44.26	≤26.18	≤**73.02**	≤54.32	≤**42.03**	≤36.57	≤39.78

the uniform-state forward process: its objective in Eq. (A8) requires evaluating the denoising term over all V vocabulary entries at every position, which adds arithmetic and memory costs that scale with V. Since UDMs are also the weaker of the two in likelihood terms (Table A9, AS1), they are dominated on both axes. We adopt MDMs throughout Section 4.

(TC3) Ermine Matches Chinchilla in Inference Speed. Time-dependent allocation alters which experts are activated at each t, not the total number of activated parameters seen by a sampling step in expectation. Consistent with this, sampling latency for the Ermine-trained model is 186.19 ms against 182.43 ms for Chinchilla, a $\sim$2% difference. Ermine thus improves quality at matched training FLOPs without shifting cost to deployment.

A.6.3 Supplementary Results

This section provides three sets of supplementary experimental results (**S1-S3**), which support the findings in Sections 4.2, 4.3, and 4.4, respectively.

(S1) Ermine Outperforms Chinchilla on Zero-shot Perplexity Evaluation. Section 4.2 compares different DLMs trained on the OpenWebText (OWT) dataset at a fixed budget of 2.89×10^{20} FLOPs. This section provides evaluation on seven commonly used textual datasets in a zero-shot setting

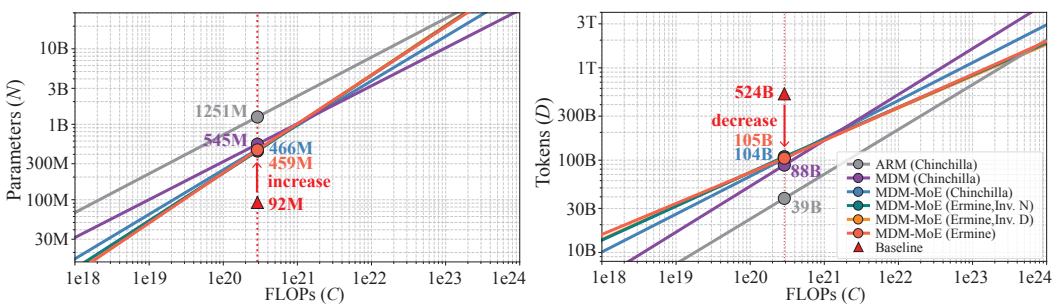

Figure A3: Compute-optimal parameter and token counts predicted by Chinchilla and Ermine. The baseline setup adopted by prior works (92M parameters, 524B tokens; red triangles) deviates from the efficient frontiers. At 2.89×10^{20} FLOPs, the laws prescribe increasing the N while decreasing D. N and D for the Ermine variants are reported as integrated values.

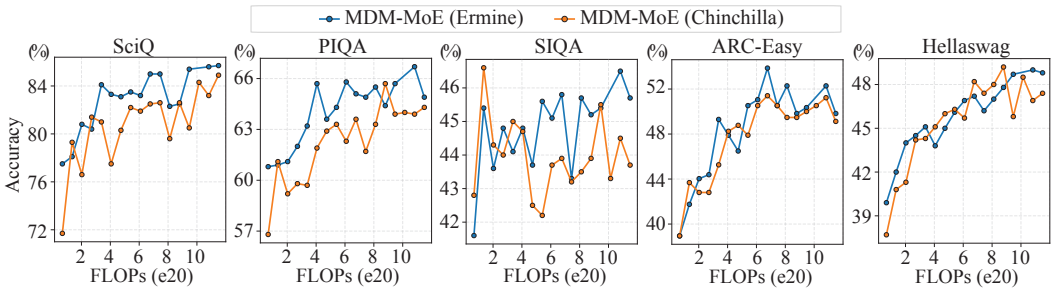

Figure A4: Zero-shot accuracy over the course of training for MDM-MoE under Ermine and Chinchilla allocation. The accuracies are evaluated on the first 1K questions with 512 MC steps.

(LAMBADA (LBD), WikiText (Wiki), Penn Treebank (PTB), LM1B, AG News (AGN), ArXiv, and PubMed (PMed)) to assess the generalizability of these models. Table A8 provides descriptions of these benchmarks, Table 2 reports the perplexity (PPL) on the OWT validation split, and Table A11 reports the zero-shot PPL.

Among order-agnostic models, every compute-optimal configuration (Ermine and Chinchilla) improves substantially over the existing non-compute-optimal DLM baselines (UDM, SEDD, MDM), indicating that compute-optimal allocation drives most of the gap. Within the compute-optimal group, the comparison of interest is MDM-MoE* (Chinchilla) against its Ermine counterparts, which share an identical architecture and differ only in how parameters and tokens are allocated across diffusion timesteps. On all seven datasets, the best Ermine variant improves over the Chinchilla allocation, with the largest margins on PTB ($84.92 \rightarrow 73.02$) and AG News ($46.85 \rightarrow 42.03$). Since none of these corpora are seen during training, the gains cannot be attributed to a closer fit of the OWT distribution alone. Instead, allocating compute according to the time-dependent scaling law yields representations that transfer across domains (see Section 4.4 for a discussion).

(S2) Loss and Accuracy of Large-Scale Pre-training. Section 4.3 compares MDM-MoE (Chinchilla) and MDM-MoE (Ermine) at 1.1×10^{21} FLOPs. This section provides supplementary results. The loss curves and the zero-shot accuracy on the five commonsense reasoning benchmarks over the course of training are shown in Figs. A5 and A4, respectively. As shown in Fig. A5, the loss of MDM-MoE (Ermine) remains lower than that of MDM-MoE (Chinchilla) throughout training, with the largest margin in the early phase and a persistent gap thereafter. Fig. A4 shows that this advantage carries over to downstream accuracy. MDM-MoE (Ermine) leads at the majority of intermediate budgets on all five benchmarks, most consistently on PIQA and SciQ, while ARC-Easy and HellaSwag are closely matched until the two curves separate late in training. The per-point fluctuations are substantial as the accuracies are estimated on the first 1K questions with 512 MC sampling steps.

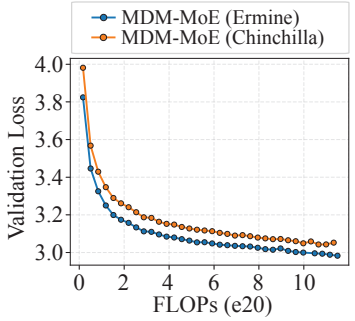

Figure A5: Validation losses of MDM-MoE (Chinchilla) and (Ermine) over the course of training.

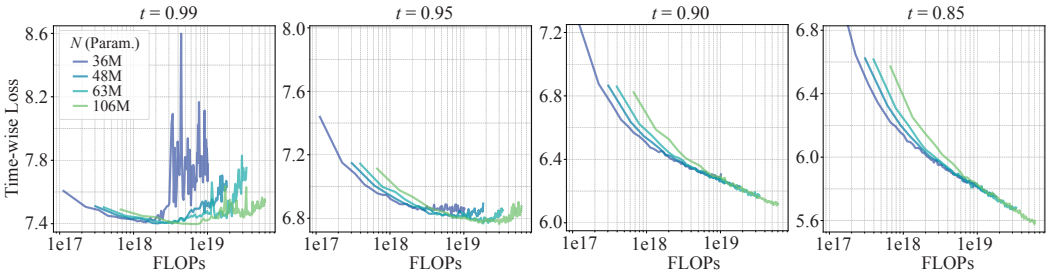

Figure A6: Time-wise validation loss of MDM-MoE on the C4 validation split. All models overfit at the noisy end ($t \to 1$), and smaller models overfit more severely.

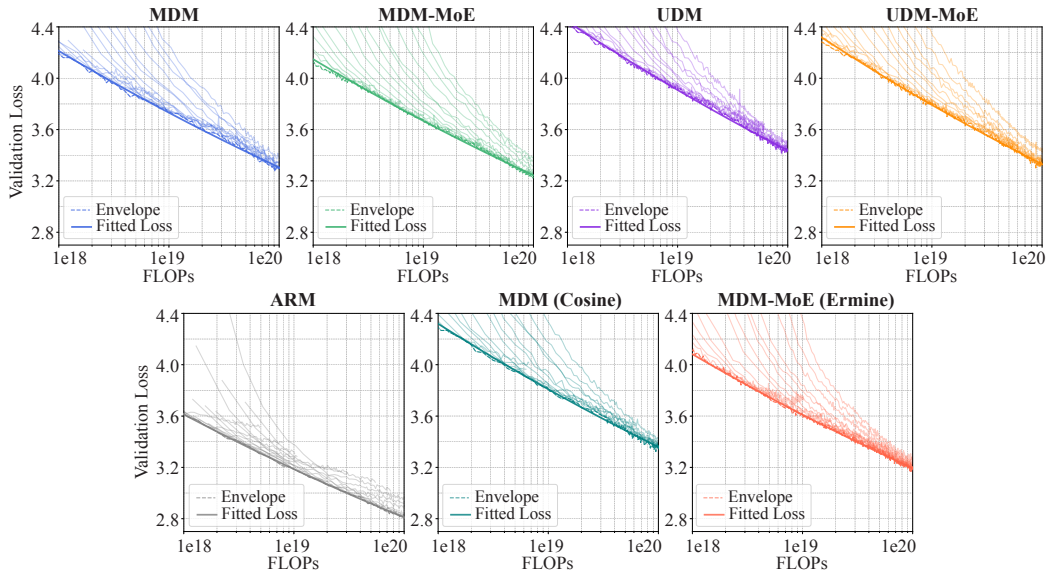

Figure A7: Validation loss versus training FLOPs for MDM, MDM-MoE, UDM, UDM-MoE, ARM, MDM (Cosine), and MDM-MoE (Ermine). All models use the Chinchilla allocation, uniform across timesteps, and a linear schedule for α_t, except that MDM-MoE (Ermine) uses the Ermine allocation and MDM (Cosine) uses a cosine schedule. Thin lines show individual runs, dashed lines their lower envelope, and solid lines the fitted power laws. Each model family follows a clean power law over 10^{18}-10^{20} FLOPs.

(S3) Capacity Improves Generalizability at Noisy End. Section 4.4 shows how the generalizability of DLMs is influenced by their time-wise scalability γ_t. This section provides further analysis of how model capacity and training token count jointly shape this behavior. Fig. A6 presents the time-wise validation loss at four timesteps near the noisy end ($t \in \{0.99, 0.95, 0.90, 0.85\}$) for models ranging from 36M to 106M parameters, and reveals two trends. First, all models overfit as $t \to 1$: the in-distribution validation loss stops decreasing and begins to rise partway through training, with the effect most pronounced at $t = 0.99$ and progressively weaker at smaller t. This corroborates the finding in Fig. 8 that the noisy end cannot absorb a uniform share of compute, and that Ermine generalizes better there precisely because C_t^* allocates less. Second, smaller models overfit earlier: the 36M model diverges first, followed by the larger ones in order of capacity. This ordering is consistent with the fitted exponents in Fig. A8 **(T3)**. As $t \to 1$, the capacity-split exponent $\frac{b_t}{a_t+b_t}$ increases while the data-split exponent $\frac{a_t}{a_t+b_t}$ decreases, meaning the compute-optimal allocation at the noisy end favors enlarging the model over extending training. A small model trained for many tokens at these timesteps is therefore far from the optimal (N_t, D_t) ratio; its capacity saturates while training continues, and the excess tokens are memorized rather than generalized.

A.6.4 Scaling Analysis

We fit the Ermine coefficients for four model families (MDM, MDM-MoE, UDM, and UDM-MoE) on the C4 dataset (see Fig. A7; first row), and report the resulting coefficient and solution compar-

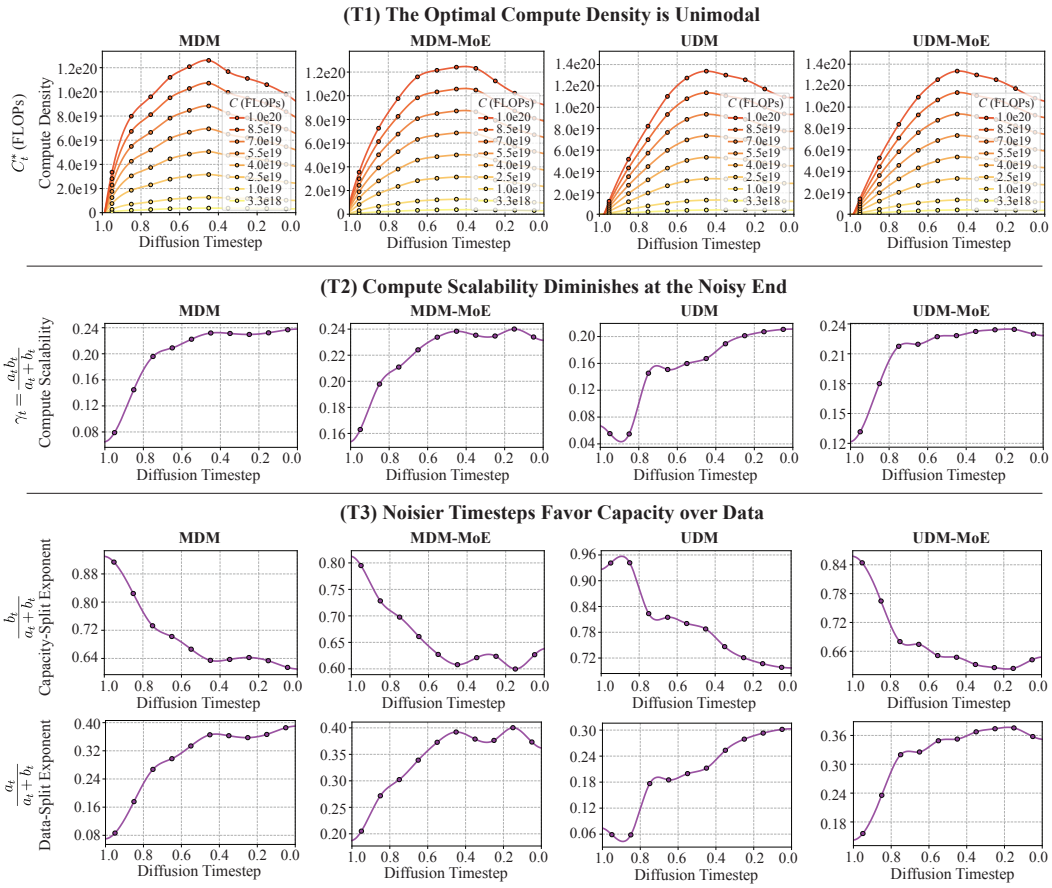

Figure A8: Three empirical trends of the Ermine solution and its coefficients across four model families (MDM, MDM-MoE, UDM, and UDM-MoE). **(T1)** The optimal compute density C_t^* is unimodal in the diffusion timestep t. **(T2)** The compute scalability $\gamma_t = \frac{a_t b_t}{a_t + b_t}$ diminishes toward the noisy end ($t \to 1$). **(T3)** Toward the noisy end, the capacity-split exponent $\frac{b_t}{a_t + b_t}$ increases while the data-split exponent $\frac{a_t}{a_t + b_t}$ decreases, indicating that noisier timesteps favor capacity over data.

isons in Fig. A8. The raw fitted coefficients $(E_t, A_t, B_t, a_t, b_t)$ are presented in Fig. A11. Three trends (**T1-T3**) hold consistently across all four families.

(T1) The Optimal Compute Density is Unimodal. The optimal compute density C_t^* is unimodal for every model family and every budget, peaking at $t \in [0.4, 0.5]$ (Fig. A8, top row). Samples in this regime contain comparable proportions of corrupted and clean tokens, making the denoising task hardest to learn and therefore the most compute-intensive. This is consistent with prior observations (Kingma & Gao, 2023; Hang et al., 2024) that the middle timesteps, where the log signal-to-noise ratio $\log \mathrm{SNR}_t \approx 0$, warrant the largest share of training samples (see Section A.3.1 for discussions of related works). For DLMs, we take $\mathrm{SNR}_t \triangleq \alpha_t/(1-\alpha_t)$ following Shi et al. (2024), since the forward kernel in Eq. (4) mixes the data distribution with weight α_t and the prior π with weight $1-\alpha_t$. Under the linear schedule $\alpha_t = 1 - t$,

$$\log \mathrm{SNR}_t = \log \frac{\alpha_t}{1-\alpha_t} = \log \frac{1-t}{t}. \tag{A16}$$

As a result, $\log \mathrm{SNR}_t = 0$ at $t = 0.5$, which sits at the point where signal and noise are balanced. To further substantiate this claim, we perform an additional scaling analysis under the cosine schedule for α_t, with results shown in Fig. A9. The cosine schedule $\alpha_t = 1 - \cos\left(\frac{\pi}{2}(1-t)\right)$ reaches $\log \mathrm{SNR}_t = 0$ at $t \approx 0.333$ (Fig. A9(**b**)), and the peak of C_t^* shifts correspondingly from $t \in [0.4, 0.5]$ to $t \in [0.2, 0.3]$ (Fig. A9(**a**)). These results support the claim that C_t^* is unimodal and peaks in the region where noisy and clean tokens are balanced.

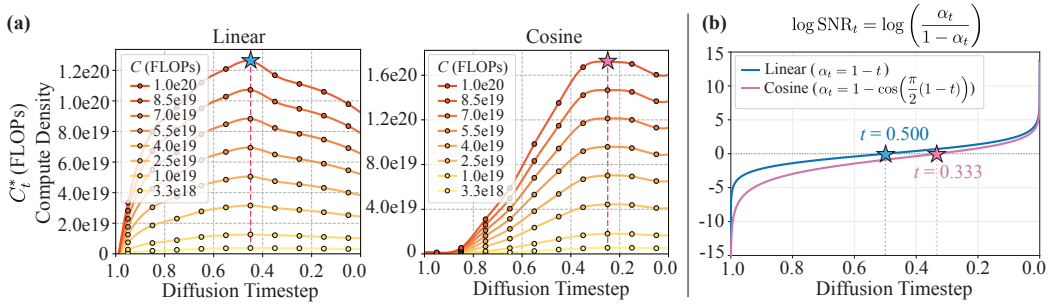

Figure A9: **(a)** Comparison between the optimal compute density C_t^* derived under the linear and cosine schedules for α_t. Stars and dashed lines mark the peaks of C_t^*, which fall in $t \in [0.4, 0.5]$ for the linear schedule and $t \in [0.2, 0.3]$ for the cosine schedule. **(b)** $\log \mathrm{SNR}_t$ (Eq. (A16)) under the two schedules, which reaches $\log \mathrm{SNR}_t = 0$ at $t = 0.5$ and $t \approx 0.333$, respectively.

(T2) Compute Scalability Diminishes at the Noisy End. The compute scalability γ_t decreases as $t \to 1$ (Fig. A8, second row), falling from above 0.20 near $t = 0$ to below 0.08 at $t = 1$ for the dense Transformers (MDM and UDM) and to below 0.17 for the MoE Transformers (MDM-MoE and UDM-MoE). Because $\hat{\ell}_t$ is decreasing in γ_t (Eq. 10), a small γ_t means that additional compute spent in the noisy regime leads to little loss reduction, and the optimal allocation accordingly assigns this region only a thin tail of C_t^* (Fig. A8).

(T3) Noisier Timesteps Favor Capacity over Data. The capacity- and data-split exponents, $\frac{b_t}{a_t+b_t}$ and $\frac{a_t}{a_t+b_t}$, follow opposite trends across t (Fig. A8, third and fourth rows). In the noisy regime ($t \to 1$), the capacity-split exponent dominates, exceeding 0.88 for the dense Transformers (MDM and UDM) and 0.80 for the MoE Transformers (MDM-MoE and UDM-MoE); compute is therefore better spent on a larger model than on longer training. The two exponents move toward each other as $t \to 0$, so the allocation becomes more balanced for cleaner samples. Section A.6.3 **(S3)** justifies this trend. The validation-loss gap between small and large models widens as $t \to 1$, confirming that capacity is what the noisy regime is sensitive to.

A.7 Limitations and Future Works

This section points out three limitations **(L1-3)** of Ermine and the corresponding directions for future work.

(L1) Data Repetition is Unmodeled. Ermine's current parameterization does not account for data repetition. Recent work shows that repetition causes scaling-law predictions to deviate in the multi-epoch regime (Muennighoff et al., 2023; Prabhudesai et al., 2025), which is where our OpenWebText experiments in Section 4.2 fall. OWT is a relatively small corpus, so the 105B training tokens we use correspond to roughly 10 epochs, outside the unique-data setting that Chinchilla-style laws assume. A formulation that models repetition explicitly would likely yield a more accurate allocation and stronger performance.

(L2) Applications to Continuous Diffusion Models. The objective in Eq. (8) requires only that the training loss decompose into an integral of time-wise losses. Traininig objectives like score-matching (Song et al., 2021) and flow-matching (Lipman et al., 2023) share this structure, so they can be substituted into Eq. (8) without modifying the formulation. Ermine is thus conceptually applicable to continuous diffusion models. This work focuses on DLMs and the text domain, and defers the discussions on continuous diffusion models to Section A.3.2. We believe extending Ermine to continuous domains such as images and videos is a promising direction for future work.

(L3) Time-dependent Post-training using Ermine. This work focuses on improving the pre-training of DLMs through compute allocation, while modern LLM training typically consists of three phases: pre-training, supervised fine-tuning (SFT), and reinforcement learning (RL). Since SFT and RL are also critical to downstream performance, a natural extension of this work is to apply Ermine to SFT and RL. Two directions merit further investigation: whether time-dependent allocation also improves post-training efficiency, and whether the resulting models reason more ef-

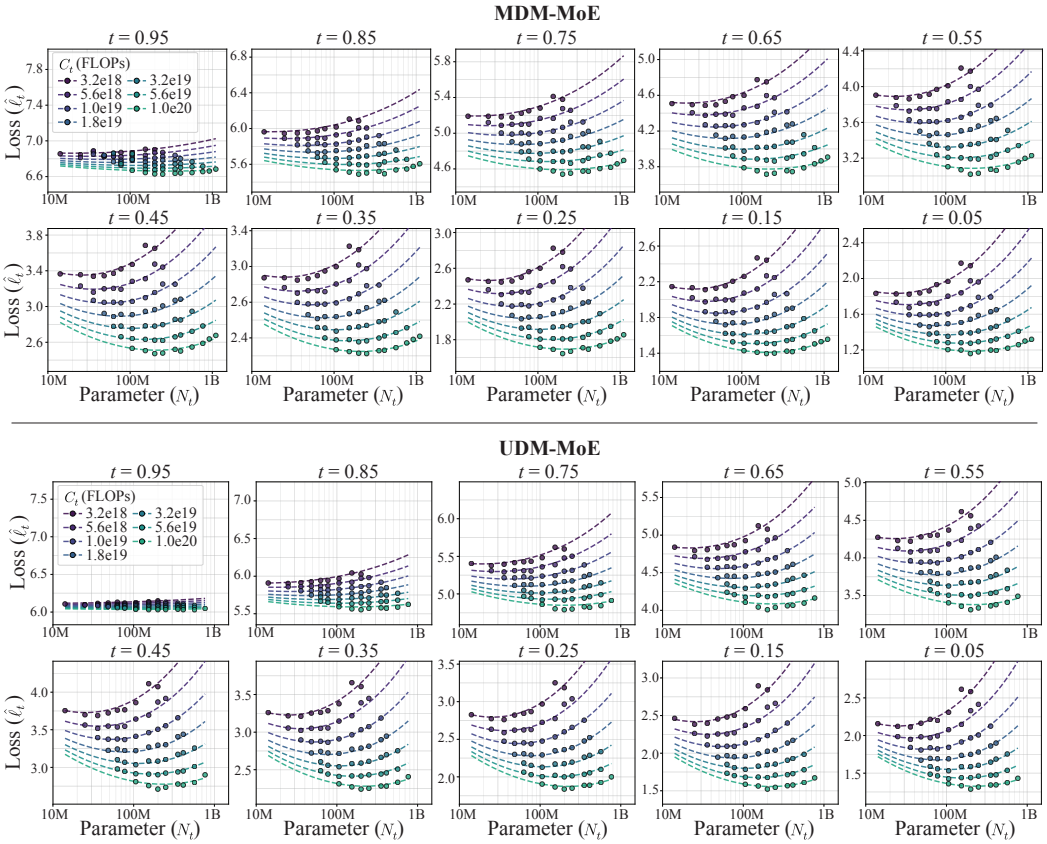

Figure A10: Time-wise scaling curves fitted by Ermine, obtained from MDM-MoE and UDM-MoE trained on C4. Each panel plots the time-wise loss $\hat{\ell}_t$ against the non-embedding activated parameters N_t under a series of C_t, with the fitted parametric curves shown as dashed lines.

ficiently. Establishing an explicit relationship between time-dependent allocation and downstream capabilities is thus a promising direction for future work.

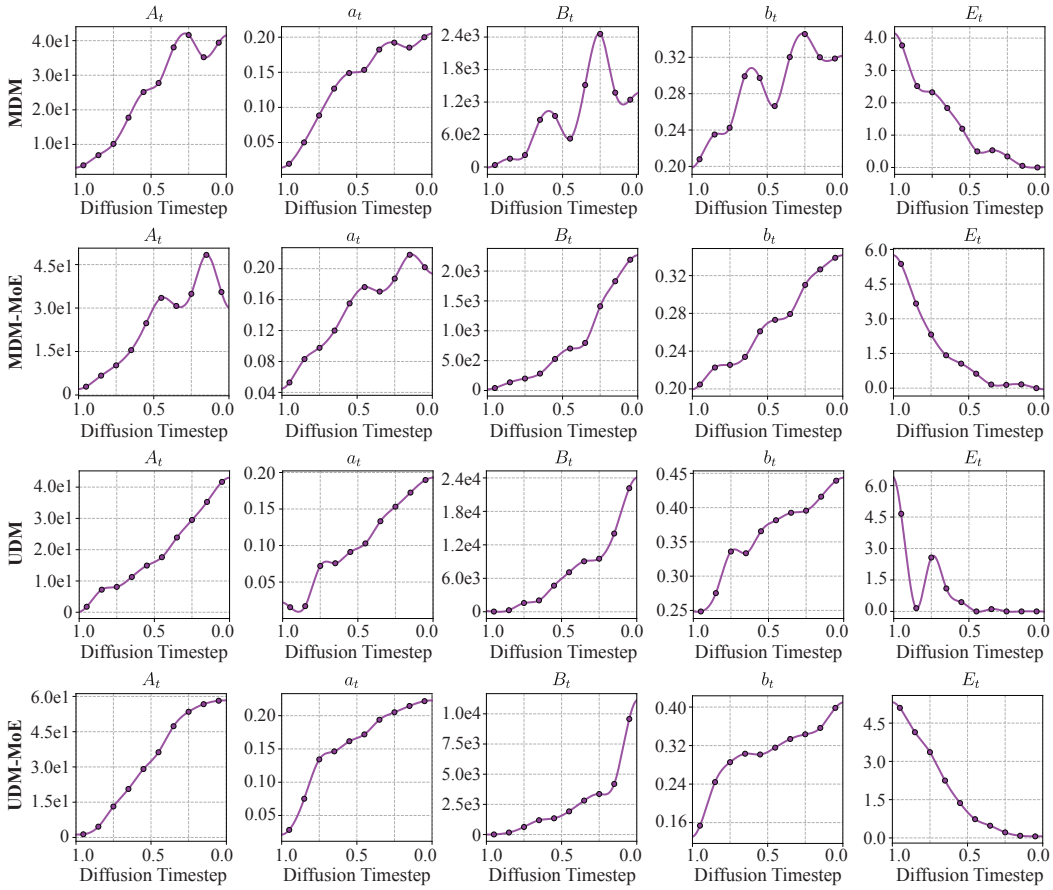

Figure A11: Ermine coefficients $(A_t, a_t, B_t, b_t, E_t)$ across diffusion timesteps for MDM, MDM-MoE, UDM, and UDM-MoE (top to bottom), obtained on the C4 dataset by minimizing the Huber objective of Eq. (2) at each timestep.

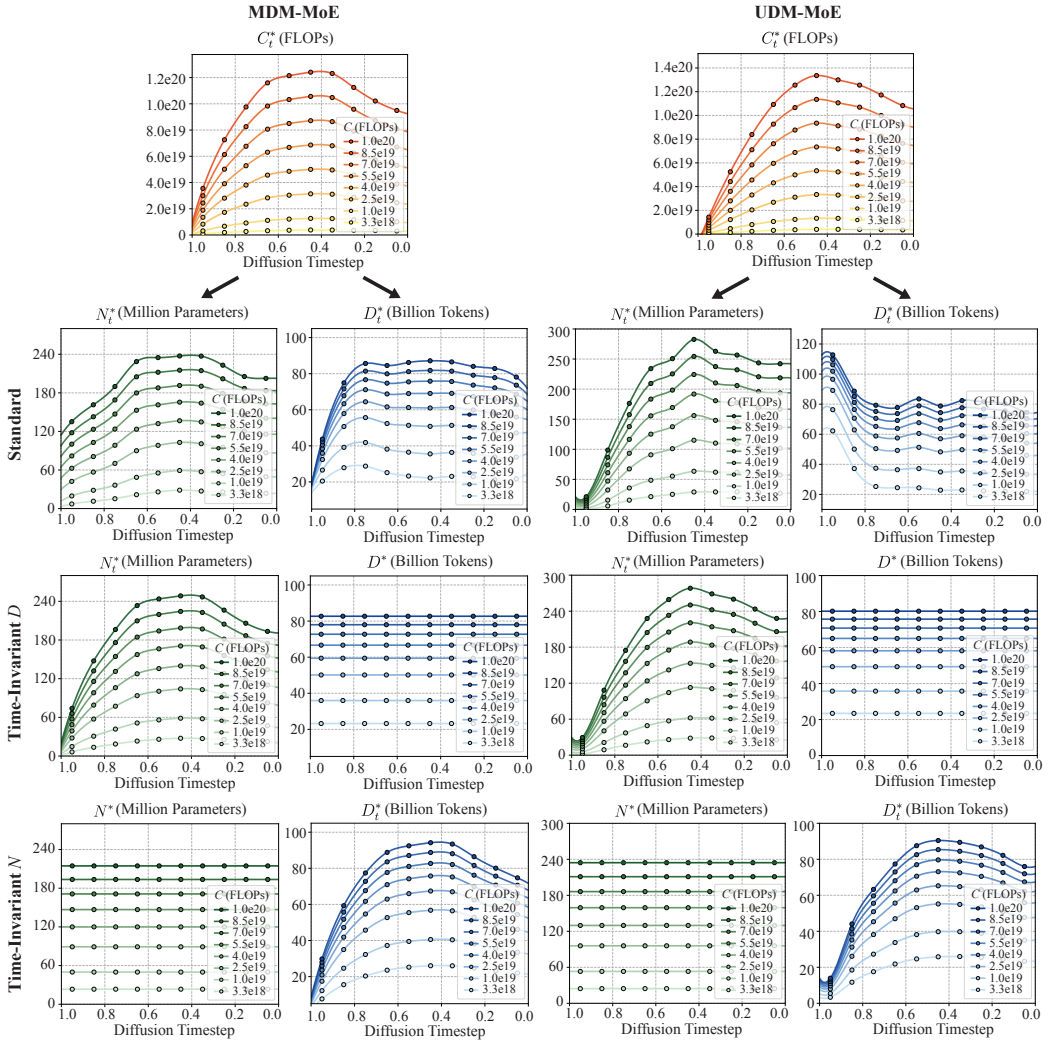

Figure A12: Decomposition of C_t^* under the standard (Eq. (10)) and time-invariant cases (Eq. (A5)) for MDM-MoE and UDM-MoE. The standard case varies both N_t^* and D_t^* across time, while the time-invariant cases hold either N or D fixed.

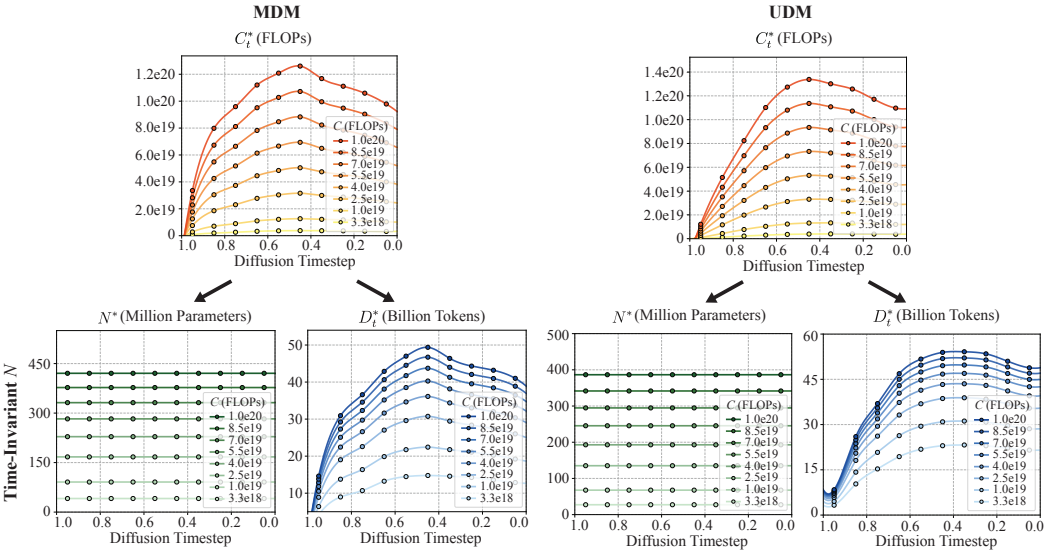

Figure A13: Decomposition of C_t^* under the time-invariant N case (Eq. (A5)) for MDM and UDM.