A DATA-CENTRIC APPROACH TO IMPROVING LLM GENERALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Simplicity bias (SB)—the tendency of gradient-based methods to prioritize dominant features—shapes how deep networks learn and generalize. Despite its demonstrated impact on generalization in CNNs, the role of SB in transformer generalization remains largely unexplored. We theoretically analyze a multi-head self-attention model and prove that Sharpness-Aware Minimization (SAM) reduces simplicity bias in transformers. Together with the empirical observation that SAM improves generalization, this suggests that reducing SB may contribute to improved generalization. Our theory further reveals that examples associated with dominant features exhibit rapid early loss drops, while examples associated with less-represented features lag behind. This makes SB observable through training dynamics and motivates a simple data-centric approach: we partition data based on early loss dynamics and amplify less-represented examples through upsampling or synthetic generation. For synthetic generation, we find that generated examples should remain sufficiently similar to their corresponding real data, achievable with diffusion LLMs. Across Llama2 (7B), Phi2 (2.7B), Llama3.2 (1B), Gemma3 (1B), and Qwen3 (0.6B), our approach consistently improves performance with AdamW and Muon, achieving a $6\times$ wall-clock speedup and $2\times$ memory reduction over SAM. Improvements reach 8.3% on mathematical reasoning, 26% on code generation, and 4.1% on instruction following.

1 INTRODUCTION

Modern deep learning theory suggests that simplicity bias (SB)—the tendency of gradient-based optimizers to learn dominant or easily represented features earlier in training—shapes how neural networks learn and generalize. SB has been shown to significantly impact generalization in CNNs (Kalimeris et al., 2019; Hu et al., 2020; Nguyen et al., 2024a), for both in-distribution (ID) and out-of-distribution (OOD) performance. Yet, despite the central role of Transformers in modern machine learning, the role of SB in transformer generalization remains largely unexplored. This motivates us to theoretically analyze SB in Transformers and ask: *Can reducing simplicity bias improve generalization in Transformers?*

We take a first step toward answering this question by theoretically analyzing multi-head linear self-attention, a tractable framework used in recent studies of transformer learning dynamics (Zhang et al., 2025; Huang et al., 2025; Javanmard et al., 2025). We characterize the early training dynamics of SAM through an ordinary differential equation (ODE), showing that attention heads sequentially learn directions of the data covariance in order of decreasing strength. We quantify simplicity bias through the entropy of feature learning times and prove that **SAM reduces simplicity bias in Transformers**. Together with SAM's improved empirical generalization, this result suggests that reducing SB may be a promising mechanism for improving generalization in Transformers. Our analysis further yields a practical prediction: examples dominated by strong, well-represented features experience rapid loss reduction early in training, while examples associated with weaker features lag behind.

This prediction turns simplicity bias from a property of optimization dynamics into an observable quantity that can be identified and controlled through the data. We exploit early training loss dynamics to partition examples according to the features they expose, identifying high-loss examples as those whose associated features have not yet been sufficiently learned. We then amplify these less-represented examples through upsampling or targeted synthetic data generation, encouraging

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.

the model to learn weaker features earlier and thereby reducing SB. For synthetic generation, we find that generated examples should remain sufficiently similar to their corresponding real examples; diffusion LLMs provide a practical way to achieve this.

Our resulting approach is simple, scalable, and optimizer-agnostic, yet consistently improves ID and OOD generalization across Llama2-7B (Touvron et al., 2023), Phi2-2.7B (Javaheripi et al., 2023), Llama3.2-1B (Dubey et al., 2024), Gemma3-1B-PT (Team Gemma et al., 2025), and Qwen3-0.6B-Base (Yang et al., 2025), with both AdamW (Loshchilov and Hutter, 2017) and Muon (Jordan et al., 2024). Improvements reach 6.1% on mathematical reasoning with upsampling and 8.3% with generation, 26% on code generation, and 4.1% on instruction following. Compared with SAM, our approach achieves a $6\times$ wall-clock speedup and $2\times$ memory reduction.

2 Related Work

Theory of Transformers. Recent theoretical work has studied Transformers in in-context learning, including convergence (Zhang et al., 2024b;a; Ren et al., 2024; Fu et al., 2024; Zhang et al., 2025), generalization (Wu et al., 2023; Mahankali et al., 2023; Duraisamy, 2024; Lu et al., 2025; Abedsoltan et al., 2024; Frei and Vardi, 2024), expressivity (Vladymyrov et al., 2024; Gatmiry et al., 2024), and loss landscapes (Mahankali et al., 2023; Li et al., 2024). Most closely related is Zhang et al. (2025), which reduces the high-dimensional training dynamics of multi-head attention to scalar ODEs and shows that attention heads sequentially learn directions of the data covariance. We build on this framework to characterize the early training dynamics of SAM and analyze its implications for SB.

Simplicity Bias. Gradient-based methods exhibit an inductive bias toward simpler solutions, which has been linked to generalization in overparameterized fully connected and convolutional networks (Belkin et al., 2019; Gunasekar et al., 2017; Hu et al., 2020; Nakkiran et al., 2021). Whether similar dynamics arise in language models trained with self-supervision has been less understood. Rende et al. (2024) provide empirical evidence that Transformers learn low-degree interactions before high-degree interactions, but the underlying dynamics remain theoretically unexplored. We provide a theoretical analysis of simplicity bias in Transformers and show that SAM reduces it, while our analysis further motivates a data-centric approach to controlling SB.

Comparison to OHEM and Curriculum Learning. Our approach differs from hard-example methods such as online hard example mining (OHEM; (Shrivastava et al., 2016)) and curriculum learning (CL; (Bengio et al., 2009)). OHEM dynamically upweights high-loss examples during training, while CL controls the ordering of examples from easy to hard. In contrast, our method identifies less-represented features from early loss dynamics and amplifies the corresponding examples once, prior to training. It targets roughly 30–45% of the data rather than a small tail of potentially difficult examples. Our approach is complementary to OHEM and CL and can be combined with both, as demonstrated in our experiments.

3 Preliminaries

We consider the widely adopted setting of a multi-head linear self-attention model for the in-context linear regression task (Garg et al., 2022; Akyürek et al., 2022; Von Oswald et al., 2023; Ahn et al., 2023; Bai et al., 2023; Zhang et al., 2025), that it is particularly well suited for studying the optimization properties of transformers (Loshchilov and Hutter, 2017; Ahn et al., 2023).

Data Model. Let $\boldsymbol{x}_i \in \mathbb{R}^d$ denote the covariates drawn from $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, and $\boldsymbol{w}_\star \in \mathbb{R}^d$ from $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$. Responses are computed as $y_i = \boldsymbol{w}_\star^\top \boldsymbol{x}_i \in \mathbb{R}$. We model in-context next-token prediction with input sequence $\{\boldsymbol{x}_1, y_1, \boldsymbol{x}_2, y_2, \ldots, \boldsymbol{x}_N, y_N, \boldsymbol{x}_q\}$, where $\boldsymbol{x}_q$ is the *query* token whose response is to be estimated. We define:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_N & \boldsymbol{x}_q \\ y_1 & y_2 & \ldots & y_N & 0 \end{bmatrix} \in \mathbb{R}^{(d+1)\times(N+1)}, \tag{1}$$

where the bottom-right entry is set to zero to indicate the unknown response y_q associated with $\boldsymbol{x}_q$. A model $f : \mathbb{R}^{(d+1)\times(N+1)} \to \mathbb{R}^{(d+1)\times(N+1)}$ predicts y_q by extracting $\hat{y}_q := f(\boldsymbol{X})_{d+1,N+1}$.

Multi-Head Linear Self-attention. For a linear self-attention model with H heads and a residual connection and $\boldsymbol{X} \in \mathbb{R}^{(d+1)\times(N+1)}$, we have:

$$\texttt{ATTN}_S(\boldsymbol{X}) = \boldsymbol{X} + \sum_{i=1}^{H} \frac{1}{N} \boldsymbol{W}_i^V \boldsymbol{X} \boldsymbol{X}^\top {\boldsymbol{W}_i^K}^\top \boldsymbol{W}_i^Q \boldsymbol{X},$$

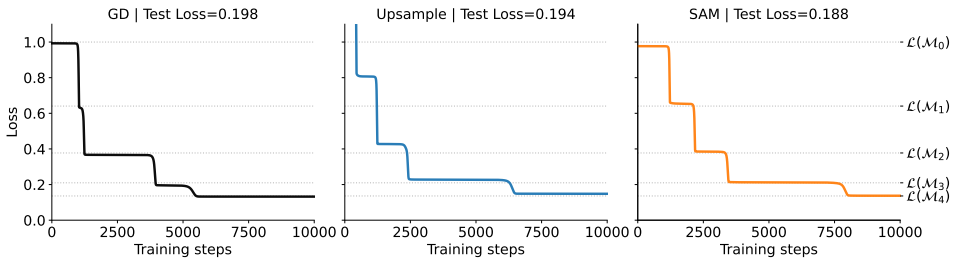

Figure 1: **Theoretical setting: Multi-head linear self-attention with rank-1 key/query matrices.** Training loss for (left) GD, (right) SAM, (middle) GD + our method. SAM and GD + our method exhibit lower simplicity bias, i.e., more uniform feature learning (drops in loss) than GD, and obtain lower test loss (reported on top of plots). In real LLMs with higher rank key-query matrices, the sharp, stage-wise loss drops of the theoretical model smooth into monotone decrease, but the underlying feature-learning sequence remains intact (see Appendix F4 in (Zhang et al., 2025) and our Figure 2).

where $\boldsymbol{W}_i^V \in \mathbb{R}^{(d+1)\times(d+1)}, \boldsymbol{W}_i^K, \boldsymbol{W}_i^Q \in \mathbb{R}^{r\times(d+1)}$ are the trainable value, key, and query matrices for head i. In practice, the model is trained with a dataset of P {tokens, query} pairs $\{\boldsymbol{X}_i, y_{q,i}\}_{i=1}^P$, where each data point can have independently sampled weight vector $\boldsymbol{w}_{\star,i} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$.

Following Zhang et al. (2025), we assume rank-one ($r = 1$) key and query matrices $\boldsymbol{W}_i^K, \boldsymbol{W}_i^Q$ for each head, which captures most of the behaviors of a general rank-R setting when training LLMs on real data (Zhang et al., 2025). Increasing the rank of key and query matrices smooths the sharp loss drops into a monotone decrease, while the core feature learning dynamics (the number of fixed points, order in which eigenvectors are learned, etc.) remain unchanged.

Lemma 1 (Equivalence to three-layer convolutional network (CNN); from §4.1 of Zhang et al. (2025))**.** *With rank-one key and query matrices and starting from a suitable initialization (Appendix A.1), the prediction from* $\texttt{ATTN}_S(\boldsymbol{X})$ *admits the following decomposition into a three-layer CNN:*

$$\texttt{ATTN}_S(\boldsymbol{X})_{d+1,N+1} = \sum_{i=1}^{H} v_i \boldsymbol{q}_i^\top \boldsymbol{K}_i \boldsymbol{z}, \quad \boldsymbol{K}_i = \begin{bmatrix} \boldsymbol{k}_i^\top & \boldsymbol{0}_d^\top & \dots & \boldsymbol{0}_d^\top \\ \boldsymbol{0}_d^\top & \boldsymbol{k}_i^\top & \dots & \boldsymbol{0}_d^\top \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0}_d^\top & \boldsymbol{0}_d^\top & \dots & \boldsymbol{k}_i^\top \end{bmatrix} \in \mathbb{R}^{d\times d^2},$$

where $\boldsymbol{z} \in \mathbb{R}^{d^2}$ *is some cubic feature map of input* $\boldsymbol{X}$*, and* $v_i \in \mathbb{R}$*,* $\boldsymbol{q}_i \in \mathbb{R}^d$*, and* $\boldsymbol{k}_i \in \mathbb{R}^d$ *contain weights from* $\boldsymbol{W}_h^K$*,* $\boldsymbol{W}_h^Q$*, and* $\boldsymbol{W}_h^V$ *respectively.*

Consequently, the summation over H *heads of* $v_i \boldsymbol{q}_i^\top \boldsymbol{K}_i \boldsymbol{z}$ *can be interpreted as applying* H *convolutional filters, with the key, query, and value weights forming the three layers.*

This three-layer CNN model is arguably the simplest nonlinear proxy for our analysis that is both sufficiently complex and tractable. More details are included in Appendix A.2.

Gradient Flow Dynamics of Early Training with GD. The linear self-attention model with learnable parameters $\boldsymbol{w}$ is trained to minimize the mean-squared loss $\mathcal{L}(\boldsymbol{w}^{(t)}) = \mathbb{E}[(y_q - \hat{y}_q)^2]$ using gradient descent (GD) at iteration t with learning rate η: $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)})$.

To reason about the training dynamics, Zhang et al. (2025) established the GD dynamics on $\texttt{ATTN}_S$ via gradient flow, which can be seen as the continuous-time limit of gradient descent as $\eta \to 0$[1]:

$$\tau \dot{\boldsymbol{W}}_{\text{GD}} = \tau \frac{\mathrm{d}\boldsymbol{W}_{\text{GD}}}{\mathrm{d}t} = -\frac{1}{2}\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{\text{GD}}} = \mathbb{E}\left[(y_q - \hat{y}_q)\frac{\partial \hat{y}_q}{\partial \boldsymbol{W}_{\text{GD}}}\right] \text{ for some time constant } \tau.$$

Feature Learning and Simplicity Bias in GD. During training, $\texttt{ATTN}_S$ activates one head at a time to learn the most prominent feature among the ones that are not learned yet. Formally, for data covariance $\boldsymbol{\Sigma} = \sum_{i=1}^d \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^\top$, where each eigenvector $\boldsymbol{e}_i$ captures a feature with strength λ_i, a head

[1]We use both matrix $\boldsymbol{W}$ and vector $\boldsymbol{w}$ to denote weights interchangeably, whichever is more appropriate in the context.

i becomes active and learns the current eigenvector $\boldsymbol{e}_h$[2] that has the largest eigenvalue not yet learned. Its key and query weights grow the fastest along the eigenvector[3].

As shown in Figure 1 (left), this results in a stage-wise training loss, with multiple abrupt drops each indicating learning a new feature by $\texttt{ATTN}_S$. As progressive learning in $\texttt{ATTN}_S$ always targets the most prominent eigenvector (easiest feature) among remaining ones, this implies the existence of SB in GD.

Global Dynamics $\Leftrightarrow$ Last-Layer v Dynamics. During the early plateau of learning $\boldsymbol{e}_{m+1}$, the layer weights approximately evolve as (Zhang et al., 2025):

$$\boldsymbol{k}_i = \boldsymbol{q}_i = v_i \boldsymbol{e}_i = \left(\lambda_i + \frac{\lambda_i + \mathrm{Tr}(\boldsymbol{\Sigma})}{N}\right)^{-1/3} \boldsymbol{e}_i \ \text{ for } 1 \le i \le m \text{ if } m \neq 0 \text{ (learned features)}; \quad (2)$$

$$\boldsymbol{k}_{m+1} = \boldsymbol{q}_{m+1} = v_{m+1}(t)\boldsymbol{e}_{m+1}; \quad \boldsymbol{k}_i = \boldsymbol{q}_i = \boldsymbol{0}, v_i = 0, \text{ for } m+2 \le i \le H. \quad (3)$$

Eq. 3 shows that only the current active head has dynamics ($v_{m+1}(t)$) and the other heads remain approximately unchanged (no dependence on t). Notably, key and query weights only depend on the value v_i and the current feature $\boldsymbol{e}_i$, which determine the norm and direction respectively. Thus, we can capture the global training dynamics only by v (last-layer) dynamics. While Eq. (2)–(3) only captures the early learning phase of each feature, Figure 1 confirms a similar behavior throughout the training.

4 Analyzing Feature Learning for SAM

In this section, we first introduce Sharpness-Aware Minimization (SAM) and then analyze its early feature learning dynamics. Finally, we build on this to compare simplicity bias of SAM vs GD.

4.1 Sharpness-Aware Minimization (SAM)

SAM performs gradient descent on weights perturbed in the gradient ascent direction, i.e. the most adversarial local perturbation. This implements a firewall against sharp minima by discouraging convergence to regions where the loss landscape could change steeply. Formally, SAM's update rules for loss $\mathcal{L}(\boldsymbol{w}^{(t)})$ at iteration t with learning rate η and perturbation strength ρ is:

$$\textbf{SAM: } \boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)} + \boldsymbol{\varepsilon}^{(t)}), \quad \boldsymbol{\varepsilon}^{(t)} = \rho \frac{\nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)})}{\|\nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)})\|_2} \ \text{ for } \rho > 0. \quad (4)$$

Note that the normalized gradient is detached from the gradient computation.

Gradient Flow for Early Training with SAM. Given a small perturbation radius ρ, we can approximate the SAM update (4) via a first-order Taylor expansion:

$$\begin{aligned} \nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)} + \boldsymbol{\varepsilon}^{(t)}) &= \nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)}) + \boldsymbol{H}^{(t)} \boldsymbol{\varepsilon}^{(t)} + O\left(\rho^2\right) \\ &\approx \left(\boldsymbol{I} + \frac{\rho}{\|\nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)})\|_2} \boldsymbol{H}^{(t)}\right) \nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)}) = (\boldsymbol{I} + \boldsymbol{P}) \nabla_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)}), \end{aligned} \quad (5)$$

where $\boldsymbol{P}$ is the "*perturbation matrix*" and $\boldsymbol{H}^{(t)} = \boldsymbol{H}(\boldsymbol{w}^{(t)}) := \nabla^2_{\boldsymbol{w}^{(t)}} \mathcal{L}(\boldsymbol{w}^{(t)})$ denotes the Hessian matrix, from which SAM "borrows curvature information" to help escape from sharp areas.

We now consider $\texttt{ATTN}_S$ trained with SAM via this approximation (5), which holds for small perturbation ρ. We study the gradient flow of loss $\mathcal{L} = \mathbb{E}[(y_q - \hat{y}_q)^2]$ under SAM. In particular,

$$\tau \dot{\boldsymbol{W}}_{\text{SAM}} = -\frac{1}{2} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \left(\boldsymbol{W} + \frac{\rho \nabla_{\boldsymbol{W}} \mathcal{L}}{\|\nabla_{\boldsymbol{W}} \mathcal{L}\|}\right) \approx (\boldsymbol{I} + \boldsymbol{P}) \tau \dot{\boldsymbol{W}}_{GD} \quad \text{where } \tau \text{ is the time constant.} \quad (6)$$

The last approximation follows from the expansion (5) and the perturbation $\boldsymbol{P}$ detached from gradient.

[2]For notational convenience, the attention heads are permuted so that the head i learns the i-th eigenvector $\boldsymbol{e}_i$

[3]Specifically, if $v_h > 0$, both $\boldsymbol{k}_h$, $\boldsymbol{q}_h$ approximately grow along $\boldsymbol{e}_h$; if $v_h < 0$, $\boldsymbol{k}_h$ and $\boldsymbol{q}_h$ grow in opposite directions, one along $\boldsymbol{e}_h$ and the other along $-\boldsymbol{e}_h$. The upcoming ansatz implicitly assumes the former.

4.2 Early Training ODE Dynamics for SAM

Next, we theoretically analyze feature learning for self-attention model trained with SAM. The following lemma shows that SAM maintains a similar distribution of fixed points to GD.

Lemma 2 (Manifolds of saddle points)**.** *Let $\boldsymbol{P}$ be the perturbation matrix induced by SAM in the first-order approximation, and $\sigma_{min}(\boldsymbol{I}+\boldsymbol{P}) > 0$ during the gradient flow. Then GD and SAM dynamics ($\dot{\boldsymbol{W}}_{SAM}$ and $\dot{\boldsymbol{W}}_{GD}$) share the same space of fixed point manifolds (see Appendix A.3) for $\texttt{ATTN}_S(\cdot)$, where $\boldsymbol{W} = \{v_i, \boldsymbol{q}_i, \boldsymbol{k}_i : i = 1, \dots, H\}$.*

Hence, under this non-degeneracy condition, the SAM flow does not introduce new saddle points or remove existing ones: any difference in feature learning for SAM vs GD as we show next, can be attributed to the optimizer-induced dynamics, rather than to a change in the "geometry of equilibria."

Next, by explicitly computing the Hessian matrices and simplifying, we derive early training dynamics as an ODE when each head starts learning.

Theorem 3 (Early Training Dynamics of SAM)**.** *Under the setting of Lemma 2, ansatz (2)-(3), and with data covariance matrix $\boldsymbol{\Sigma} = \sum_{i=1}^{d} \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^\top$ indexed by decreasing distinct bounded eigenvalues, the early training dynamics of GD and first-order SAM expansion (5) on $\texttt{ATTN}_S(\boldsymbol{X})$ when learning the i-th feature/eigenvector ($i \in [\min\{H, d\}]$) is approximately captured by the one scalar-variable ODEs:*

$$\textbf{\textit{SAM (ours)}} : \tau \dot{v}_i = \lambda_i^2 v_i^2 - \frac{2\rho}{\sqrt{3}} \lambda_i^2 v_i, \quad \textbf{\textit{GD (Zhang et al., 2024b)}} : \tau \dot{v}_i = \lambda_i^2 v_i^2 \tag{7}$$

under sufficiently small weight initializations.

The proof can be found in Appendix C. Clearly, SAM introduces a perturbation-dependent term that slows down learning across all the features. This slow-down factor turns out to be non-uniform and alters how feature learning is appointed, as we formalize in the next section.

4.3 SAM Has Lower Simplicity Bias than GD

We now build upon the ODE results and prove that during early training, SAM effectively uniformizes feature learning (i.e. reduces SB) in our multi-head self-attention model. We first establish a proxy for quantifying SB, and show that the times to complete early feature learning form a more uniform sequence for SAM compared to that of GD.

Entropy as a Proxy for Simplicity Bias. Let $t_1, \dots, t_M$ denote the (feature-wise) times taken by the model to learn M features. Lower SB means that learning is uniform, where a small subset of features does not consume most of the time. To measure this evenness in a scale-free way, we sum-normalize the time sequence as $p_i = t_i / \sum_j t_j, 1 \le i \le M$. Viewing this as discrete probability masses, we quantify SB via its entropy:

$$\textbf{Entropy}\left(\{t_i\}_{i=1}^M\right) = -\sum_{i=1}^{M} \frac{t_i}{\sum_j t_j} \log\left(\frac{t_i}{\sum_j t_j}\right) = -\sum_{i=1}^{M} p_i \log p_i. \tag{8}$$

A higher entropy indicates a discrete distribution closer to a uniform one and smaller SB.

Quantifying SAM's Simplicity Bias in Early Training. Theorem 3 describes an early-training regime where the terms in Eq. (7) govern the evolution of $v_i(t)$. We can then take proper integrals to obtain the learning times of each v_i during early training.

Note that the final learned weight v_i in Eq. (2), given a large number of input tokens N, is:

$$v_i \boldsymbol{e}_i = \left(\lambda_i + \frac{\lambda_i + \mathrm{Tr}(\boldsymbol{\Sigma})}{N}\right)^{-1/3} \boldsymbol{e}_i = O(\lambda_i^{-1/3}) \boldsymbol{e}_i.$$

This motivates our final assumption that early training learns each feature minimally.

Assumption 4 (Minimal feature learning in early training)**.** For each feature, the evolution of weight v_i admits a small constant c s.t. $v_i = c\lambda_i^{-1/3}$ around the end of early training for *GD and SAM*.

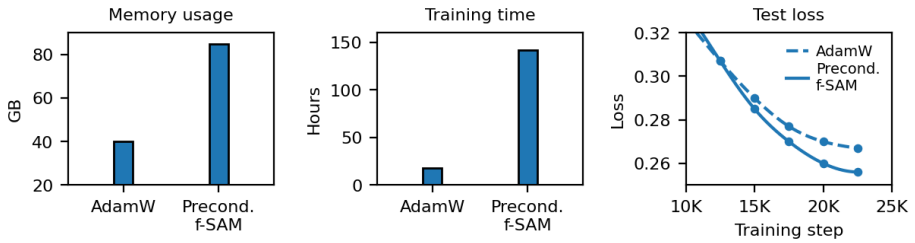

Figure 2: **Comparison of preconditioned f-SAM and AdamW when fine-tuning Phi2-2.7B on MathInstruct.** f-SAM achieves lower evaluation loss compared to AdamW but incurs 2x GPU memory and 6x training time, and is prohibitively expensive for training even medium-sized LLMs (we could not finish training after six days on 4× NVIDIA A40 GPUs, and hence report validation loss).

Assumption 4 implicitly provides integral bounds on v_i so that we can directly integrate the inverse of the ODEs from Theorem 3 and obtain two time sequences. Specifically, starting with some initialization v_0, the time of early training for head i, by Theorem 3, is:

$$t_i^{GD} = \int_{v_0}^{c\lambda_i^{-\frac{1}{3}}} \frac{dt}{dv_i}\, dv_i = \int_{v_0}^{c\lambda_i^{-\frac{1}{3}}} \frac{\tau}{\lambda_i^2 v_i^2}\, dv_i.$$

A similar sequence can be defined for SAM. Now we can use majorization theory (Marshall et al., 1979) to quantify Simplicity Bias via Entropy. More details can be found in Appendix C.4. The following theorem presents our key result regarding the uniformity in SAM feature learning.

Theorem 5 (SAM has lower simplicity bias than GD)**.** *In the setting of Theorem 3 and Assumption 4, suppose* $\texttt{ATTN}_S(\boldsymbol{X})$ *learns* M *features in total starting from the same small initializations for both SAM and GD. Each* $v_i > 0$ *starts at* ε_i *and reaches some* $c\lambda_i^{-1/3}$ *around the end of early training,* $i \in [M]$*. Taking* $\varepsilon = \max\{\varepsilon_i\}_{i=1}^M$*, we integrate over the early training phase shared for all features to get the times:*

$$t_i^{GD} := \int_{\varepsilon}^{c\lambda_i^{-1/3}} \frac{\tau}{\lambda_i^2 v_i^2}\, dv_i, \qquad t_i^{SAM} := \int_{\varepsilon}^{c\lambda_i^{-1/3}} \frac{\tau}{\lambda_i^2 v_i^2 - \frac{2\rho}{\sqrt{3}}\lambda_i^2 v_i}\, dv_i$$

Then as long as $\rho < \sqrt{3}\varepsilon_i/2$*, we have that* $\textbf{Entropy}\left(\left\{t_i^{SAM}\right\}_{i=1}^M\right) > \textbf{Entropy}\left(\left\{t_i^{GD}\right\}_{i=1}^M\right)$.

That is, SAM results in lower simplicity bias early in training compared to GD.

Figure 1 (right) empirically verifies that the drops in training loss are more uniform globally and under more general Gaussian initializations (no $v > 0$ restriction). Notably, while our analysis only considers early training, in practice similar results hold throughout training.

Remark. Theorem 5 shows that SAM reduces simplicity bias by making feature learning more homogeneous across directions early in training. Together with the empirical observation that SAM improves generalization, this suggests that reducing simplicity bias may contribute to improved generalization in Transformer models.

Despite its superior generalization (Singh et al., 2025), SAM is prohibitively expensive for LLMs. Figure 2 shows that when fine-tuning Phi-2 with 2.7B parameters, SAM yields a lower test loss in expense of over 2× memory and 6× training time. This motivates our next section, which proposes a data-centric method to reduce SB during training with gradient-based optimizers, e.g., AdamW, Muon.

5 Data-centric Approach to Reduce Simplicity Bias

The theoretical analysis establishes that SAM reduces simplicity bias (SB) by making feature learning more uniform. This suggests a complementary data-centric strategy: rather than modifying the optimizer, we modify the training data to amplify less-represented features.

A key prediction of our theory is that features are learned in order of decreasing eigenvalue strength: examples dominated by well-represented features exhibit rapid early loss reductions, while those

containing less-represented features lag behind. In the theoretical model, eigenvalue strength captures how strongly each feature is represented in the data distribution, with stronger features learned earlier. In real language data, examples similarly contain multiple lexical, syntactic, stylistic, or semantic features. Although these strengths are not directly observable, their effect is reflected in per-example loss trajectories, motivating them as a proxy for identifying less-represented features.

Identifying Less-Represented Features. We train a smaller proxy model for a short period (e.g., $1/3$ of an epoch) and record each example's loss at a few checkpoints. We then cluster the loss trajectories into two groups and identify the high-loss cluster as examples associated with less-represented features. This lightweight procedure is performed once as a preprocessing step.

Amplifying Less-Represented Features. We amplify selected examples through duplication or source-conditioned synthetic generation. Duplication increases the frequency of corresponding features but also amplifies example-specific noise. Synthetic augmentation can instead vary this noise while preserving the underlying features, provided generated examples remain sufficiently similar to their sources. Naive LLM rephrasing lacks such control and can alter the underlying features, resulting in poor performance. In contrast, diffusion LLMs control the degree of change, generating sufficiently similar examples that preserve the original features while varying example-specific noise.

The resulting intervention encourages standard gradient-based optimization to allocate more updates to less-represented features, approximating the more uniform feature-learning dynamics associated with reduced SB. We empirically validate this effect across models, datasets, and optimizers in §7.

Computational Efficiency. Unlike SAM, which incurs additional computation and memory overhead at every optimization step, our data intervention is performed once before training. The small proxy model is trained only briefly, resulting in an overall ~$1.3\times$ training overhead for upsampling, after which standard first-order optimization is used without additional overhead. In our experiments, our method reduces memory usage by $2\times$ and achieves up to $8\times$ wall-clock speedup over SAM.

We provide the complete pseudocode for our approach in Appendix D.1 (Algorithm 1).

6 Experiments

Here, we describe the training and evaluation recipes to evaluate our data-centric approach.

Datasets. We evaluate across three structurally distinct domains. For mathematical reasoning, we use MathInstruct (Yue et al., 2023), a diverse corpus of over 262K instruction-response pairs from 13 heterogeneous sources. For code generation, we use the Python subset of Magicoder-OSS-Instruct-75K (Wei et al., 2024), filtered to 50–500-token solutions. For instruction following, we use Alpaca-cleaned (Taori et al., 2023), a curated set of 52K examples spanning creative writing, brainstorming, explanation, and general conversation.

Finetuning. We consider five pretrained models spanning 0.6B–7B parameters: Qwen3-0.6B-Base (Yang et al., 2025), Llama3.2-1B (Dubey et al., 2024), Gemma3-1B-PT (Team Gemma et al., 2025), Phi2-2.7B (Javaheripi et al., 2023), and Llama2-7B (Touvron et al., 2023). Following Nguyen et al. (2024b), we fine-tune all models with LoRA (Hu et al., 2022), using AdamW as the default optimizer and Muon to test optimizer-agnostic behavior (Appendix D.2).

We use Pythia-70M (Biderman et al., 2023) as a lightweight proxy. We record per-example losses at five checkpoints during $1/3$ epoch of proxy fine-tuning and cluster the trajectories using k-means.

Evaluation All models are evaluated zero-shot with greedy decoding (maximum 2,048 tokens). For math, we report accuracy on GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), NumGLUE (Mishra et al., 2022), SVAMP (Patel et al., 2021), DeepMind (Davies et al., 2021), and MMLU-Math (Hendrycks et al., 2020). For code, we report pass@1 on HumanEval(+) (Chen et al., 2021) and MBPP(+) (Austin et al., 2021) with prompts under 512 tokens. For instruction following, we report MT-Bench (Zheng et al., 2023) scores on writing, roleplay, reasoning, extraction, STEM, and humanities, evaluated by GPT-4 Turbo.

Synthetic Data Generation. We use LLaDA-8B-Base (Nie et al., 2025) to reconstruct partially masked problem statements without access to their original answers, followed by GPT-OSS-20B (OpenAI, 2025) to generate supervision and, when needed, distractor options. Candidates undergo task-specific checks; after three failed attempts, we duplicate the original example (details in Appendix G).

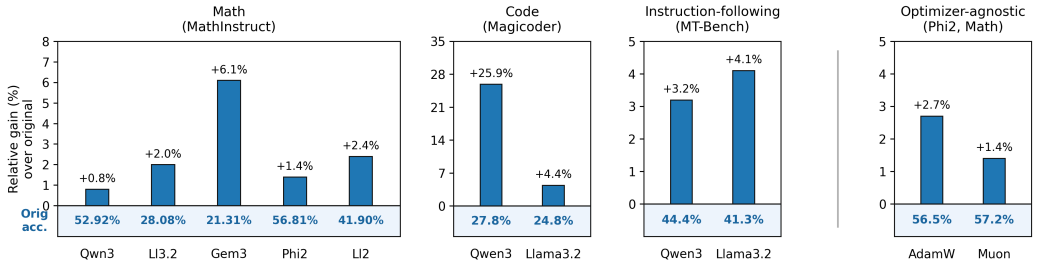

Figure 4: **Performance gains from targeted upsampling across models, domains, and optimizers.** Each model is evaluated zero-shot after standard fine-tuning and targeted upsampling. Our method yields consistent gains across model families and domains under AdamW and remains optimizer-agnostic under Muon (right). Standard fine-tuning accuracies are shown in **blue** below the zero line for Qwen3-0.6B-Base; Llama3.2-1B; Gemma3-1B-PT; Phi2-2.7B; Llama2-7B.

7 Results

Figures 3 and 4 report relative accuracy gains of our method over standard finetuning across all domains and models (dataset-level results are in Appendix E).

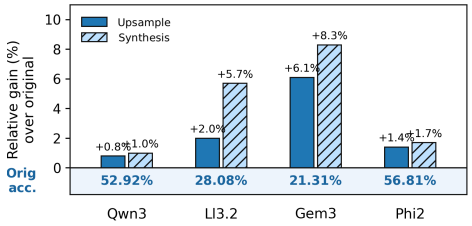

Figure 3: Synthetic augmentation further boosts performance on MathInstruct.

Math, Code, and Instruction-Following. Our method identifies 63% of MathInstruct, 45% of Magicoder, and 28% of Alpaca-cleaned as examples containing less-represented features. A qualitative inspection of the MathInstruct clusters shows that the well-represented cluster is dominated by repetitive multiple-choice arithmetic templates, while the less-represented cluster captures less frequent patterns such as Python-code rationales and open-ended symbolic derivations (see Appendix H). Upsampling these examples consistently improves MathInstruct performance across five model families (0.6B–7B) under AdamW, with relative gains of up to 6.1% across six benchmarks. For code generation, we observe gains of up to 26% on HumanEval(+) and MBPP(+), while MT-Bench scores also improve on open-ended instruction following. Finally, consistent improvements with both AdamW and Muon (Figure 4, rightmost panel) demonstrate that our approach is optimizer-agnostic.

Effect of Model Scale and Pretraining. Targeted upsampling benefits depend on both model capacity and pretraining exposure. Models with sufficient capacity but less exposure to relevant reasoning features show the strongest gains, while those with limited capacity or extensive exposure benefit less. Llama2-7B nevertheless shows consistent improvements, indicating that reducing simplicity bias remains effective at larger scales (See Appendix E.1 for details).

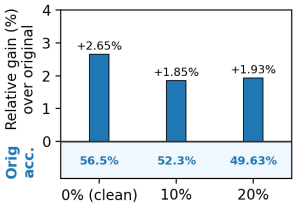

Figure 5: Robustness to label noise (Phi2, Math).

Synthetic Augmentation. As shown in Figure 3, LLaDA-based synthetic augmentation outperforms exact duplication across all four model families on MathInstruct, achieving relative accuracy gains of up to 8.3% over finetuning on the original dataset. Additional results in Tables 11 and 12.

Evidence of Reduced SB. In real LLMs, higher loss on well-represented examples indicates lower SB, reflecting reduced reliance on simple features. We verify this for Llama3.2-1B on MathInstruct: upsampling increases loss on well-represented examples by 25% and decreases loss on less-represented examples by 78% relative to standard fine-tuning, directly confirming reduced SB.

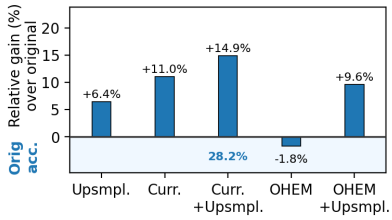

Figure 6: Upsampling with curriculum learning and OHEM (Llama3.2, Math).

Robustness to Label Noise. *Early* in training, the model focuses on learning general prompt structure rather than target outputs; consequently, mislabeled examples do not exhibit distinctively high early loss and are not selectively concentrated in the less-represented cluster. To verify this, we apply random label swapping to 10% and 20% of MathInstruct and finetune Phi2-2.7B under each noise level. As

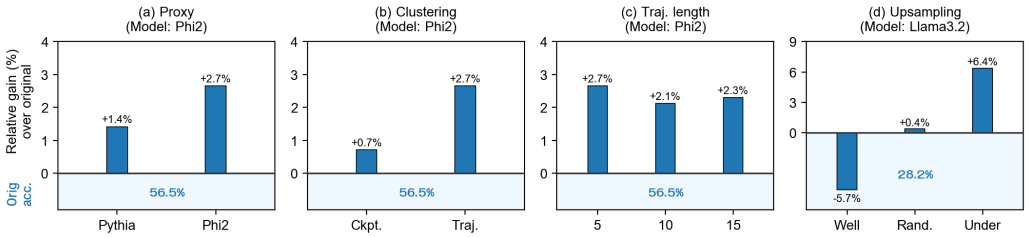

Figure 7: **Ablations.** (a) Pythia-70M performs comparably to the target model as a proxy. (b) Full loss trajectories are competitive with the final checkpoint alone. (c) Performance is stable across 5, 10, and 15 checkpoints. (d) Upsampling less-represented examples gives the strongest generalization, while upsampling well-represented examples degrades performance.

shown in Figure 5, our method consistently improves over standard finetuning across all noise levels. More importantly, under 10% label swapping, both clusters contain approximately 10% mislabeled examples, confirming that our method does not disproportionately upsample noisy examples.

Computational Efficiency. Our data-centric approach decouples feature reweighting from the optimization loop and incurs only ~$1.4\times$ the cost of standard AdamW. Proxy profiling requires $\approx 0.3\%$ of baseline compute, while upsampling the selected cluster ($\approx 38\%$ of the dataset) increases fine-tuning steps by $1.38\times$. Overall, our approach achieves over $4\times$ wall-clock speedup compared with SAM while remaining compatible with standard first-order optimizers.

Stacking with Curriculum Learning and OHEM. While curriculum learning (CL) and OHEM target example ordering and high-loss outliers, respectively, our method reshapes the global training distribution ($\approx 40\%$ of the corpus) to mitigate simplicity bias. Thus, the approaches are complementary: combining UP with CL yields consistent gains over either alone, while OHEM+UP substantially improves over standard fine-tuning and recovers the performance of OHEM alone.

Other Confounding Factors. In Appendix E.2, we show that cluster membership is not explained by sequence length and reliably distinguishes less- from well-represented examples.

7.1 Ablations

Fig 7 reports ablations for finetuning on MathInstruct with AdamW; detailed results in Appendix F.

Choice of Proxy Model. Fig 7(a) shows that using Pythia-70M achieves comparable performance as a proxy for Phi2-2.7B, despite identifying 28% fewer examples for upsampling. The smaller proxy learns well-represented features faster, yielding a smaller less-represented cluster that nevertheless remains sufficient for consistent gains, demonstrating robustness to proxy choice.

Clustering Strategy. As shown in Fig 7(b), clustering with full loss trajectories outperforms clustering by final-checkpoint loss for Phi2-2.7B, with mixed results across other models (Appendix E). This confirms our theoretical results and the necessity of *early* loss trajectories.

Trajectory Length. Performance remains stable across 5, 10, and 15 proxy checkpoints ($1/3$–1 epoch), with the less-represented cluster spanning 38–63% of the data (Fig 7c). Together with Figure 7(b), this shows that the first $1/3$ epoch already captures the relevant signal.

Upsampling Strategy. Compute-matched results (Fig 7(d) show that random upsampling provides no benefit over standard fine-tuning, while upsampling well-represented examples degrades performance. In contrast, upsampling less-represented examples yields the strongest generalization, confirming that gains come from targeting less-represented features rather than increasing sample count.

8 Conclusion

We presented a theoretical and data-centric study of simplicity bias in transformers, showing that SAM reduces simplicity bias and that early loss dynamics provide a practical signal for identifying less-represented features. Building on this insight, we proposed a simple approach that amplifies such features through targeted upsampling or synthetic generation. Across models, domains, and optimizers, our approach consistently improves generalization while reducing the computational and memory overhead of SAM; demonstrating that simplicity bias can be controlled through training data, offering an efficient alternative to optimizer-level interventions for improving LLM generalization.

AI-Use Statement

We used generative AI tools in several aspects of this work. GPT-OSS-20B was used to generate and verify synthetic examples in our data augmentation experiments (Figure 3, Appendix G), with prompts documented in Appendix G.3. GPT-4 Turbo was used as the evaluator for MT-Bench, following its standard evaluation protocol. We also used LLM-based coding assistants to implement standard baselines, with all implementations reviewed and verified by the authors. Generative AI was not used to develop research ideas, formulate theoretical derivations or claims, or produce experimental results. We also used generative AI tools to assist with manuscript writing, including grammar, phrasing, and clarity; all AI-assisted content was reviewed and edited by the authors.

Reproducibility Statement

To facilitate reproducibility, we provide complete pseudocode for our data-centric method in Algorithm 1 and describe the full training and evaluation setup in §6. This includes the datasets, pretrained models, evaluation benchmarks, metrics, decoding strategy, and proxy-model procedure. Complete hyperparameter configurations, including LoRA ranks, learning rates, sequence lengths, and optimizer settings, are provided in Appendix D.2, with additional ablation settings described in §7.1. All datasets and pretrained models used in our experiments are publicly available. We use deterministic greedy decoding for evaluation, eliminating stochasticity at inference time, and additionally evaluate sensitivity to training stochasticity through multi-seed experiments for representative model, dataset, and optimizer combinations (see Tables 6 and 8). Together with consistent improvements across three domains, five model families, two optimizers, and eleven benchmarks, these results provide evidence that our findings are robust to experimental variation. To support further verification and extension, we will release the full codebase upon acceptance.

References

Amirhesam Abedsoltan, Adityanarayanan Radhakrishnan, Jingfeng Wu, and Mikhail Belkin. Context-scaling versus task-scaling in in-context learning. *arXiv preprint arXiv:2410.12783*, 2024.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36:45614–45650, 2023.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Alexander Atanasov, Blake Bordelon, and Cengiz Pehlevan. Neural networks as kernel learners: The silent alignment effect. *arXiv preprint arXiv:2111.00034*, 2021.

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

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36:57125–57211, 2023.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

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 *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

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

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

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600(7887):70–74, 2021.

Simon S Du, Wei Hu, and Jason D Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. *Advances in neural information processing systems*, 31, 2018.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

Karthik Duraisamy. Finite sample analysis and bounds of generalization error of gradient descent in in-context linear regression. *arXiv preprint arXiv:2405.02462*, 2024.

Spencer Frei and Gal Vardi. Trained transformer classifiers generalize and exhibit benign overfitting in-context. *arXiv preprint arXiv:2410.01774*, 2024.

Deqing Fu, Tian-Qi Chen, Robin Jia, and Vatsal Sharan. Transformers learn to achieve second-order convergence rates for in-context linear regression. *Advances in Neural Information Processing Systems*, 37:98675–98716, 2024.

Kenji Fukumizu. Effect of batch learning in multilayer neural networks. In *International Conference on Neural Information Processing*, 1998. URL https://api.semanticscholar.org/CorpusID:605683.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J Reddi, Stefanie Jegelka, and Sanjiv Kumar. On the role of depth and looping for in-context learning with task diversity. *arXiv preprint arXiv:2410.21698*, 2024.

Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. *Advances in neural information processing systems*, 30, 2017.

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

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Wei Hu, Lechao Xiao, Ben Adlam, and Jeffrey Pennington. The surprising simplicity of the early-time learning dynamics of neural networks. *Advances in Neural Information Processing Systems*, 33: 17116–17128, 2020.

Jianhao Huang, Zixuan Wang, and Jason D. Lee. Transformers learn to implement multi-step gradient descent with chain of thought. In *The Thirteenth International Conference on Learning Representations*, 2025.

Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 1(3):3, 2023.

Adel Javanmard, Baharan Mirzasoleiman, and Vahab Mirrokni. Understanding the role of training data in test-time scaling, 2025. URL https://arxiv.org/abs/2510.03605.

Ziwei Ji and Matus Telgarsky. Gradient descent aligns the layers of deep linear networks. *arXiv preprint arXiv:1810.02032*, 2018.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. https://kellerjordan.github.io/posts/muon/, 2024.

Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. Sgd on neural networks learns functions of increasing complexity. *Advances in neural information processing systems*, 32, 2019.

Yingcong Li, Ankit S Rawat, and Samet Oymak. Fine-grained analysis of in-context linear estimation: Data, architecture, and beyond. *Advances in Neural Information Processing Systems*, 37:138324–138364, 2024.

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

Yue M Lu, Mary Letey, Jacob A Zavatone-Veth, Anindita Maiti, and Cengiz Pehlevan. Asymptotic theory of in-context learning by linear attention. *Proceedings of the National Academy of Sciences*, 122(28):e2502599122, 2025.

Arvind Mahankali, Tatsunori B Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv preprint arXiv:2307.03576*, 2023.

Albert W Marshall, Ingram Olkin, and Barry C Arnold. *Inequalities: theory of majorization and its applications*. Springer New York, NY, 1979.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks. *arXiv preprint arXiv:2204.05660*, 2022.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Dang Nguyen, Paymon Haddad, Eric Gan, and Baharan Mirzasoleiman. Changing the training data distribution to reduce simplicity bias improves in-distribution generalization. *Advances in Neural Information Processing Systems*, 37:68854–68896, 2024a.

Dang Nguyen, Wenhan Yang, Rathul Anand, Yu Yang, and Baharan Mirzasoleiman. Mini-batch coresets for memory-efficient language model training on data mixtures. *arXiv preprint arXiv:2407.19580*, 2024b.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. *arXiv preprint arXiv:2508.10925*, 2025.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

Yunwei Ren, Zixuan Wang, and Jason D Lee. Learning and transferring sparse contextual bigrams with linear transformers. *Advances in Neural Information Processing Systems*, 37:20304–20357, 2024.

Riccardo Rende, Federica Gerace, Alessandro Laio, and Sebastian Goldt. A distributional simplicity bias in the learning dynamics of transformers. *Advances in Neural Information Processing Systems*, 37:96207–96228, 2024.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Andrew M Saxe, James L McClelland, and Surya Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116(23): 11537–11546, 2019.

Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Sidak Pal Singh, Hossein Mobahi, Atish Agarwala, and Yann Dauphin. Avoiding spurious sharpness minimization broadens applicability of sam. *arXiv preprint arXiv:2502.02407*, 2025.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Aishwarya Team Gemma, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Max Vladymyrov, Johannes Von Oswald, Mark Sandler, and Rong Ge. Linear transformers are versatile in-context learners. *Advances in Neural Information Processing Systems*, 37:48784–48809, 2024.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: empowering code generation with oss-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Jingfeng Wu, Difan Zou, Zixiang Chen, Vladimir Braverman, Quanquan Gu, and Peter L Bartlett. How many pretraining tasks are needed for in-context learning of linear regression? *arXiv preprint arXiv:2310.08391*, 2023.

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

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55, 2024a.

Ruiqi Zhang, Jingfeng Wu, and Peter Bartlett. In-context learning of a linear transformer block: Benefits of the mlp component and one-step gd initialization. *Advances in Neural Information Processing Systems*, 37:18310–18361, 2024b.

Yedi Zhang, Aaditya K Singh, Peter E Latham, and Andrew Saxe. Training dynamics of in-context learning in linear attention. *arXiv preprint arXiv:2501.16265*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

APPENDIX

A Equivalence between Stylized Attention and Neural Networks **16**

A.1 Initialization . . . 16

A.2 Proof Techniques (Zhang et al., 2024a) . . . 16

A.3 Manifolds of Fixed Points . . . 17

B Preliminary Investigation into Merged Key-Query Matrix **18**

B.1 Necessary Gradients & Hessians . . . 18

B.2 Proof of Lemma 2 and Stability of Saddle Landscape . . . 18

B.3 Training Dynamics ODE for White Covariance . . . 19

B.4 Escape Time from Saddles for General Covariance . . . 21

B.5 Helper Lemmas . . . 23

C Proofs of Theorems for $\texttt{ATTN}_S$ **25**

C.1 Preliminary and Our Hessian Extension . . . 25

C.2 Learning the First Dominant Eigenvector: Proof of Theorem 3 Part 1 . . . 26

C.3 Saddle-to-Saddle Learning: Proof of Theorem 3 Part 2 . . . 27

C.4 Learning Entropy: Proof of Theorem 5 . . . 28

C.5 Majorization Theory: Helper Lemmas . . . 29

D Implementation Details **31**

D.1 Pseudocode . . . 31

D.2 Hyperparameters . . . 31

E Detailed Experimental Results **33**

E.1 Effect of Model Scale and Pretraining . . . 33

E.2 Confound Analysis . . . 34

F Detailed Ablation Results **35**

G Synthetic Augmentation **37**

G.1 Source-Conditioned Generation with LLaDA . . . 37

G.2 Gradient-Space Analysis . . . 37

G.2.1 Setup and Conditional Decomposition . . . 37

G.2.2 Residual Aggregation and Comparison with Duplication . . . 38

G.2.3 Retaining the Original Example . . . 38

G.3 Details of generation . . . 39

H Qualitative Characterization of Trajectory Clusters **43**

I Broader Impacts and Limitations **44**

A Equivalence between Stylized Attention and Neural Networks

A.1 Initialization

$\boldsymbol{W}_i^V = \begin{bmatrix} * & * \\ \boldsymbol{0} & v_i \end{bmatrix}, \boldsymbol{W}_i^K = \begin{bmatrix} \boldsymbol{k}_i^\top & 0 \end{bmatrix}, \boldsymbol{W}_i^Q = \begin{bmatrix} \boldsymbol{q}_i^\top & * \end{bmatrix}$, where * indicates values that do not matter; the weights initialized to zero are not required to achieve global minimum loss on the in-context linear regression task (Ahn et al., 2023; Zhang et al., 2024a; 2025).

A.2 Proof Techniques (Zhang et al., 2024a)

In this section, we briefly discuss the proof techniques from (Zhang et al., 2024a) and reiterate some results regarding the reduction of multi-head self-attention models into neural networks. First, recall the separate key-query configuration:

$$\texttt{ATTN}_S(\boldsymbol{X}) = \boldsymbol{X} + \sum_{i=1}^{H} \frac{1}{N} \boldsymbol{W}_i^V \boldsymbol{X} \boldsymbol{X}^\top \boldsymbol{W}_i^{K\top} \boldsymbol{W}_i^Q \boldsymbol{X},$$

$$\text{with } \boldsymbol{W}_i^V = \begin{bmatrix} * & * \\ \boldsymbol{v}_i & v_i \end{bmatrix}, \boldsymbol{W}_i^K = \begin{bmatrix} \boldsymbol{k}_i^\top & k_i \end{bmatrix}, \boldsymbol{W}_i^Q = \begin{bmatrix} \boldsymbol{q}_i^\top & q_i \end{bmatrix}.$$

Here (*) denotes sub-matrices with appropriate dimensions whose values do not matter at all. Given $\hat{y}_q = \texttt{ATTN}_S(\boldsymbol{X})_{d+1,N+1}$ the predicted response for the query and the loss $\mathcal{L} = \mathbb{E}[(y_q - \hat{y}_q)^2]$, Appendix F.1 of (Zhang et al., 2024b) justifies the choice of zero blocks by showing that the gradients w.r.t k_i and $\boldsymbol{v}_i$ equal 0, meaning that they remain at initializations throughout training. Hence, we directly set the initialization to be 0 for simplicity.

Now we can plug in the weights and it follows that:

$$\begin{aligned}
\texttt{ATTN}_S(\boldsymbol{X}) &= \boldsymbol{X} + \sum_{i=1}^{H} \frac{1}{N} \boldsymbol{W}_i^V \boldsymbol{X} \boldsymbol{X}^\top \boldsymbol{W}_i^{K\top} \boldsymbol{W}_i^Q \boldsymbol{X} \\
&= \boldsymbol{X} + \frac{1}{N} \sum_{i=1}^{H} \begin{bmatrix} * & * \\ \boldsymbol{0} & v_i \end{bmatrix} \begin{bmatrix} \sum_{i=1}^N \boldsymbol{x}_i \boldsymbol{x}_i^\top + \boldsymbol{x}_q \boldsymbol{x}_q^\top & \sum_{i=1}^N y_i \boldsymbol{x}_i \\ \sum_{i=1}^N y_i \boldsymbol{x}_i^\top & \sum_{i=1}^N y_i^2 \end{bmatrix} \begin{bmatrix} \boldsymbol{k}_i \\ 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_i^\top & * \end{bmatrix} \boldsymbol{X} \\
&= \boldsymbol{X} + \frac{1}{N} \sum_{i=1}^{H} \begin{bmatrix} * & * \\ v_i \sum_{i=1}^N y_i \boldsymbol{x}_i^\top & v_i \sum_{i=1}^N y_i^2 \end{bmatrix} \begin{bmatrix} \boldsymbol{k}_i \boldsymbol{q}_i^\top & * \\ \boldsymbol{0} & 0 \end{bmatrix} \boldsymbol{X} \\
&= \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_N & \boldsymbol{x}_q \\ y_1 & y_2 & \dots & y_N & 0 \end{bmatrix} + \frac{1}{N} \sum_{i=1}^{H} \begin{bmatrix} * & * \\ v_i \sum_{i=1}^N y_i \boldsymbol{x}_i^\top \boldsymbol{k}_i \boldsymbol{q}_i^\top & * \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_N & \boldsymbol{x}_q \\ y_1 & y_2 & \dots & y_N & 0 \end{bmatrix}.
\end{aligned}$$

We take the bottom right entry and have:

$$\hat{y}_q = \texttt{ATTN}_S(\boldsymbol{X})_{d+1,N+1} = 0 + \frac{1}{N} \sum_{i=1}^{H} \sum_{i=1}^{N} y_i \boldsymbol{x}_i^\top v_i \boldsymbol{k}_i \boldsymbol{q}_i^\top \boldsymbol{x}_q.$$

Let $\texttt{vec}(\cdot)$ denote the column-wise flattening of a matrix into a column vector. Define the following cubic map of the input $\boldsymbol{X}$, and algebra shows that:

$$\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{X}) = \texttt{vec}\left(\frac{1}{N} \sum_{i=1}^{N} y_i \boldsymbol{x}_i \boldsymbol{x}_q^\top \right) \in \mathbb{R}^{d^2} \implies \tag{9}$$

$$\hat{y}_q = \sum_{i=1}^{H} v_i \boldsymbol{q}_i^\top \boldsymbol{K}_i \boldsymbol{z}, \quad \text{with} \quad \boldsymbol{K}_i = \begin{bmatrix} \boldsymbol{k}_i^\top & \boldsymbol{0}_d^\top & \dots & \boldsymbol{0}_d^\top \\ \boldsymbol{0}_d^\top & \boldsymbol{k}_i^\top & \dots & \boldsymbol{0}_d^\top \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0}_d^\top & \boldsymbol{0}_d^\top & \dots & \boldsymbol{k}_i^\top \end{bmatrix} \in \mathbb{R}^{d \times d^2}.$$

It is also common in theory to consider a simpler model, where the key and query matrices are merged into one single matrix that aims to simplify the analysis further:

$$\texttt{ATTN}_M(\boldsymbol{X}) = \boldsymbol{X} + \sum_{i=1}^{H} \frac{1}{N} \boldsymbol{W}_i^V \boldsymbol{X} \boldsymbol{X}^\top \boldsymbol{W}_i^{KQ} \boldsymbol{W}_i^Q \boldsymbol{X}, \quad \text{where } \boldsymbol{W}_i^V = \begin{bmatrix} * & * \\ \boldsymbol{v}_i & v_i \end{bmatrix}, \boldsymbol{W}_i^{KQ} = \begin{bmatrix} \boldsymbol{U}_i & * \\ \boldsymbol{u}_i^\top & * \end{bmatrix}.$$

While we focus on the separate configuration, we provide some discussion in the appendices. First, by repeating the similar analysis as above, (Zhang et al., 2025) show that we can initialize $\boldsymbol{v}_i, \boldsymbol{u}_i = \mathbf{0}$ and subsequently, some algebra shows that

$$\texttt{ATTN}_M(\boldsymbol{X})_{d+1,N+1} = \boldsymbol{w}_2^\top \boldsymbol{W}_1 \boldsymbol{z}, \quad \text{with } \boldsymbol{w}_2 = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_H \end{bmatrix} \in \mathbb{R}^H, \ \boldsymbol{W}_1 = \begin{bmatrix} \texttt{vec}(\boldsymbol{U}_1) \\ \texttt{vec}(\boldsymbol{U}_2) \\ \vdots \\ \texttt{vec}(\boldsymbol{U}_H) \end{bmatrix},$$

which is a two-layer fully-connected linear neural network. This makes sense intuitively, as merging key and query together loses information and can no longer admit a third layer. In the following section, we provide the manifolds of fixed points in the training dynamics of each.

A.3 Manifolds of Fixed Points

We directly reiterate the conclusions from Zhang et al. (2025), in particular Section 3.2 and Appendix E.3, with slightly modifed notations.

For $\texttt{ATTN}_M(\boldsymbol{X})_{d+1,N+1}$, there are two manifolds of fixed points, corresponding to vanishing initialization and final learning:

$$\mathcal{M}_0 = \{\boldsymbol{w}_2 = \mathbf{0}, \boldsymbol{W}_1 = \mathbf{0}\}, \quad \mathcal{M}_* = \left\{\boldsymbol{w}_2, \boldsymbol{W}_1 : \boldsymbol{w}_2^\top \boldsymbol{W}_1 = \mathbb{E}\left[y_q \boldsymbol{z}^\top\right] \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]^{-1}\right\}.$$

Here learning is achieved when the model weights move from small initializations (around the zero saddle) to the other stable manifold, resulting in a sharp drop in training loss.

For $\texttt{ATTN}_S(\boldsymbol{X})_{d+1,N+1}$ instead, there are 2^d manifolds of fixed points. Let the covariance $\boldsymbol{\Sigma} = \sum_{i=1}^d \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^\top$ ordered by the eigenvalues. A manifold exists for each subset of learned eigenvectors with indices $\mathcal{S}_m \subseteq \{1, 2, \ldots, d\}$ where $m = 0, \ldots, d$, $|\mathcal{S}_m| = m$, and the manifold of fixed points for this set is given by:

$$\begin{aligned} \mathcal{M}(\mathcal{S}_m) = \Big\{ & v_i, \boldsymbol{q}_i, \boldsymbol{K}_i : i = 1, \ldots, H; \text{ such that the following three conditions hold:} \\ & (1) \sum_{i=1}^H v_i \boldsymbol{k}_i \boldsymbol{q}_i^\top = \sum_{j \in \mathcal{S}_m} \left(\lambda_j + \frac{\lambda_j + \operatorname{Tr}(\boldsymbol{\Sigma})}{N}\right)^{-1} \boldsymbol{e}_j \boldsymbol{e}_j^\top; \\ & (2)\ v_i \neq 0 \implies \text{both } \boldsymbol{k}_i,\ \boldsymbol{q}_i \in \operatorname{Span}\left\{\boldsymbol{e}_j\right\}_{j \in \mathcal{S}_m}; \\ & (3)\ v_i = 0 \implies \text{at least one of } \boldsymbol{k}_i,\ \boldsymbol{q}_i \in \operatorname{Span}\left\{\boldsymbol{e}_j\right\}_{j \in \mathcal{S}_m} \Big\}. \end{aligned}$$

The precise derivation of these conditions come from an equality constrained optimization problem. However, the key takeaway here is that the ansatz (2)–(3) satisfies the above conditions and serves as a reasonable educated guess to facilitate analyzing the highly nonlinear training dynamics. Here learning is done progressively by learning one eigenvector at a time, in the order of decreasing eigenvalues. We observe saddle-to-saddle learning dynamics, where the model moves from a saddle in $\mathcal{M}(\text{top } i \text{ eigenvector indices})$ to a saddle in $\mathcal{M}(\text{top } i+1 \text{ eigenvector indices})$.

Remark. The latter observation of progressively learning the eigenvalues from largest to smallest reflects the idea of simplicity bias, i.e., easy (more obvious) features are learned before difficult (less intuitive) ones. However, a practical concern is that if the model gets stuck at one saddle point for too long, it might reach convergence before getting to learn more complex features. As a result, reducing simplicity bias can help the model "escape" earlier and explore more.

B Preliminary Investigation into Merged Key-Query Matrix

In this part of the appendix, we provide an attempt to study the simpler setting of merged key-query configuration $\texttt{ATTN}_M$ with a SAM variant suing similar techniques. From Appendix A.3, this model has two manifolds of fixed points, one around small initializations, and escaping into the other one corresponds to the drop in training loss. We establish early training ODEs and insights on the dynamics, similar to the $\texttt{ATTN}_S$ results.

B.1 Necessary Gradients & Hessians

Recall that we study the expected squared loss for the predicted label of a new query $\mathcal{L} = \mathbb{E}[(y_q - \hat{y}_q)^2]$. In particular, for joint key-query matrix, y_q is the output from a fully connected two-layer neural network, and the loss becomes:

$$\mathcal{L} = \mathbb{E}\left[\left(y_q - \boldsymbol{w}_2^\top \boldsymbol{W}_1 \boldsymbol{z}\right)^2\right].$$

This gives us the following gradient and Hessian with respect to $\boldsymbol{w}_2$ and $\boldsymbol{W}1$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_2} = \nabla_{\boldsymbol{w}_2}\mathcal{L} = -2\mathbb{E}\left[\left(y_q - \boldsymbol{w}_2^\top \boldsymbol{W}_1 \boldsymbol{z}\right)\left(\boldsymbol{W}_1 \boldsymbol{z}\right)\right] = -2\boldsymbol{W}_1\left(\mathbb{E}\left[y_q \boldsymbol{z}^\top\right] - \boldsymbol{w}_2^\top \boldsymbol{W}_1 \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\right)^\top.$$
$$\boldsymbol{H}(\boldsymbol{w}_2) = \nabla^2_{\boldsymbol{w}_2}\mathcal{L} = 2\boldsymbol{W}_1 \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\boldsymbol{W}_1^\top.$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_1} = \nabla_{\boldsymbol{W}_1}\mathcal{L} = -2\mathbb{E}\left[\left(y_q - \boldsymbol{w}_2^\top \boldsymbol{W}_1 \boldsymbol{z}\right)\left(\boldsymbol{w}_2 \boldsymbol{z}^\top\right)\right] = -2\boldsymbol{w}_2\left(\mathbb{E}\left[y_q \boldsymbol{z}^\top\right] - \boldsymbol{w}_2^\top \boldsymbol{W}_1 \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\right).$$

For Hessian matrix, it makes more sense to consider $\boldsymbol{w}_1 = \texttt{vec}(\boldsymbol{W}_1) \in \mathbb{R}^{d^2}$ flattened into a vector. Let $\otimes$ denote the Kronecker matrix product. We then have

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \texttt{vec}(\boldsymbol{W}_1)} = \nabla_{\boldsymbol{w}_1}\mathcal{L} &= -2\texttt{vec}\left(\boldsymbol{w}_2 \mathbb{E}\left[y_q \boldsymbol{z}^\top\right]\right) - \texttt{vec}\left(\boldsymbol{w}_2 \boldsymbol{w}_2^\top \boldsymbol{W}_1 \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\right) \\
&= -2\texttt{vec}\left(\boldsymbol{w}_2 \mathbb{E}\left[y_q \boldsymbol{z}^\top\right]\right) - 2\left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \left(\boldsymbol{w}_2 \boldsymbol{w}_2^\top\right)\right)\texttt{vec}\left(\boldsymbol{W}_1\right) \\
&= -2\texttt{vec}\left(\boldsymbol{w}_2 \mathbb{E}\left[y_q \boldsymbol{z}^\top\right]\right) - 2\left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \left(\boldsymbol{w}_2 \boldsymbol{w}_2^\top\right)\right)\boldsymbol{w}_1. \\
\boldsymbol{H}(\texttt{vec}(\boldsymbol{W}_1)) = \nabla^2_{\boldsymbol{w}_1}\mathcal{L} &= 2\left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \left(\boldsymbol{w}_2 \boldsymbol{w}_2^\top\right)\right).
\end{aligned}$$

To simplify the analysis, we study the merged key-query configuration under layer-wise SAM defined below. We argue in the subsequent Section B.3 that it derives similar insights.

Definition 6 (Global vs. layer-wise SAM regimes)**.** For model weights $\boldsymbol{w} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k)$ of k groups/layers, we consider the following two SAM regimes:

Global SAM: All the parameters are updated at once via Equation 4.

Layer-wise SAM: Each layer of parameters is updated separately via k SAMs:

$$\tau \dot{\boldsymbol{w}}_i = -\frac{1}{2}\nabla_{\boldsymbol{w}_i}\mathcal{L}\left(\boldsymbol{w}_i + \frac{\rho \nabla_{\boldsymbol{w}_i}\mathcal{L}}{\|\nabla_{\boldsymbol{w}_i}\mathcal{L}\|_2}\right).$$

Note that in the main paper, we consider $\texttt{ATTN}_S$ with the practical global SAM (More precisely, since the ansatz guarantees that only the current active head has dynamics, the other heads remain unchanged and do not contribute any gradient. Hence, layer-wise SAM on the active head/layer is equivalent to global SAM).

B.2 Proof of Lemma 2 and Stability of Saddle Landscape

Under the assumption that $\sigma_{\min}\left(\boldsymbol{I} + \boldsymbol{P}\right) > \boldsymbol{0}$ during gradient flow i.e. the perturbation never cancels out the raw gradient, the lemma statement is trivial from the approximation 5:

$$\tau \dot{\boldsymbol{W}}_{\text{SAM}} \approx (\boldsymbol{I} + \boldsymbol{P})\tau \dot{\boldsymbol{W}}_{GD},$$

where $\boldsymbol{I} + \boldsymbol{P}$ is full rank. Theoretically, this allows us to isolate the observation of reduced SB to the SAM optimizer, instead of distribution of saddles or significant change of loss landscape.

B.3 Training Dynamics ODE for White Covariance

A similar conservation law (ansatz) among the weights can be established for the two-layer fully connected neural network equivalence $\texttt{ATTN}_M$ (see (Fukumizu, 1998; Du et al., 2018; Saxe et al., 2019; Ji and Telgarsky, 2018; Zhang et al., 2025)) that derives from vanishing initializations:

$$\boldsymbol{w}_2\boldsymbol{w}_2^\top - \boldsymbol{W}_1\boldsymbol{W}_1^\top \approx \boldsymbol{0} \implies \|\boldsymbol{w}_2\|_2 \approx \|\boldsymbol{W}_1\|_F = \|\texttt{vec}(\boldsymbol{W}_1)\|_2.$$

Suppose that the perturbation radius ρ is sufficiently small in the sense that it preserves this conservation law. Since the two layers of weights maintain equal norms and hence evolve similarly, under the additional ansatz that their gradient norms $\|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2$ and $\|\nabla_{\boldsymbol{W}_1}\mathcal{L}\|_F$ are of the same order, global SAM update now depends on the perturbation: let $\boldsymbol{w} = (\texttt{vec}(\boldsymbol{W}_1)^\top, \boldsymbol{w}_2^\top)^\top = (\boldsymbol{w}_1^\top, \boldsymbol{w}_2^\top)^\top$,

$$\begin{aligned}\nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{w} + \rho\frac{\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})}{\|\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})\|_2}\right) &= \nabla_{\boldsymbol{w}}\mathcal{L}\left(\begin{bmatrix}\boldsymbol{w}_1\\ \boldsymbol{w}_2\end{bmatrix} + \frac{\rho}{\sqrt{\|\nabla_{\boldsymbol{w}_1}\mathcal{L}\|_2^2 + \|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2^2}}\begin{bmatrix}\nabla_{\boldsymbol{w}_1}\mathcal{L}\\ \nabla_{\boldsymbol{w}_2}\mathcal{L}\end{bmatrix}\right)\\ &= \nabla_{\boldsymbol{w}}\mathcal{L}\left(\begin{bmatrix}\boldsymbol{w}_1\\ \boldsymbol{w}_2\end{bmatrix} + \frac{\rho}{\sqrt{C\|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2^2 + \|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2^2}}\begin{bmatrix}\nabla_{\boldsymbol{w}_1}\mathcal{L}\\ \nabla_{\boldsymbol{w}_2}\mathcal{L}\end{bmatrix}\right)\\ &\qquad \text{for constant } C.\end{aligned}$$

We focus on the update on $\boldsymbol{w}_2$:

$$\nabla_{\boldsymbol{w}_2}\mathcal{L}\left(\boldsymbol{w}_2 + \frac{\rho}{\sqrt{(C+1)\|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2^2}}\nabla_{\boldsymbol{w}_2}\mathcal{L}\right) = \nabla_{\boldsymbol{w}_2}\mathcal{L}\left(\boldsymbol{w}_2 + \frac{\hat{\rho}\nabla_{\boldsymbol{w}_2}\mathcal{L}}{\|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2}\boldsymbol{w}_2\right), \text{ where } \hat{\rho} = \frac{\rho}{\sqrt{C+1}}.$$

A similar argument holds for $\boldsymbol{w}_1$. Hence, layer-wise SAM becomes similar to global SAM up to a scaled parameter $\hat{\rho}$. This would partially justify how layer-wise SAM can potentially yield similar insights.

We now focus on the setting of white covariance i.e. $\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^\top] = \alpha\boldsymbol{I}_{d^2}$; recall that $\boldsymbol{z}$ (Eq. 9) is a cubic feature map on the input $\boldsymbol{X}$. Similarly, due to the weight conservation ansatz, the evolution of $\boldsymbol{w}_2$ captures global dynamics.

Theorem 7 (SAM accelerates training for merged KQ)**.** *Under sufficiently small initializations for $\texttt{ATTN}_M$, suppose that SAM perturbation preserves the weight conservation $\boldsymbol{w}_2\boldsymbol{w}_2^\top - \boldsymbol{W}_1\boldsymbol{W}_1^\top \approx 0$ and can be well approximated by the first-order expansion 5. Let $s = \boldsymbol{w}_2^\top\boldsymbol{w}_2 = \|\boldsymbol{W}_1\|_F^2$, $\gamma = \|\mathbb{E}[y_q\boldsymbol{z}]\|_2$, and $\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^\top] = \alpha\boldsymbol{I}$ i.e. white input data covariance. Then the evolution of the weights $s(t)$ when we perform layer-wise SAM on $\boldsymbol{w}_2$ satisfies:*

$$\frac{t}{\tau} = \frac{1}{2\gamma}\ln\frac{s(t)}{s(0)} + \frac{1}{\alpha\sqrt{\rho^2 + \frac{4\gamma}{\alpha}}}\left(\frac{1}{r_-}\ln\frac{\sqrt{s(t)} - r_-}{\sqrt{s(0)} - r_-} - \frac{1}{r_+}\ln\frac{\sqrt{s(t)} - r_+}{\sqrt{s(0)} - r_+}\right),$$
$$\text{with } r_\pm = \frac{\rho \pm \sqrt{\rho^2 + \frac{4\gamma}{\alpha}}}{2}.$$

In particular, this equation is the implicit closed-form solution to the following SAM dynamics (when $\rho = 0$, the first term recovers GD only dynamics in Zhang et al. (2025)):

$$\tau\dot{s} = 2s(\gamma - \alpha s) + 2\rho\alpha s^{3/2}, \quad \text{where } \gamma - \alpha s > 0.$$

By convention, we let $\tau\dot{s} = 0$ when $s = \gamma/\alpha$ i.e. the weight increases and reaches equilibrium.

Note that the dynamics has an unstable equilibrium at $s = 0$ and a stable one at $s = \gamma/\alpha$. In the continuous flow, small initializations start near the former and move to the latter via this ODE, where the training loss sharply decreases. In particular, the SAM dynamics implements a boosting term $2\rho\alpha s^{3/2}$ to allow the weights to evolve faster.

Proof. According to (Saxe et al., 2013), the conservation law implies:

$$\boldsymbol{W}_1 = \boldsymbol{w}_2\frac{\mathbb{E}[y_q\boldsymbol{z}^\top]}{\|\mathbb{E}[y_q\boldsymbol{z}]\|_2} = \boldsymbol{w}_2\boldsymbol{m}^\top, \quad \text{where } \boldsymbol{m} = \frac{\mathbb{E}[y_q\boldsymbol{z}^\top]}{\|\mathbb{E}[y_q\boldsymbol{z}]\|_2} \text{ is a unit vector.} \tag{10}$$

which gives rise to the following ODE that describes the GD dynamics in Zhang et al. (2025):

$$\tau(\dot{\boldsymbol{w}}_2)_{\text{GD}} = \boldsymbol{w}_2(\|\mathbb{E}[y_q\boldsymbol{z}]\|_2 - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2) = -\frac{1}{2}\nabla_{\boldsymbol{w}_2}\mathcal{L}.$$

With the existing theory, we are now ready to derive the approximate SAM dynamics. We start with the perturbation matrix $\boldsymbol{P}$ using formulas in Appendix B.3:

$$\begin{aligned}
\boldsymbol{P} = \frac{\rho}{\|\nabla_{\boldsymbol{w}_2}\mathcal{L}\|_2}\boldsymbol{H}(\boldsymbol{w}_2) &= \frac{\rho}{2\left\|\boldsymbol{W}_1\left(\mathbb{E}\left[y_q\boldsymbol{z}^\top\right] - \boldsymbol{w}_2^\top\boldsymbol{W}_1\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\right)^\top\right\|_2}\left(2\boldsymbol{W}_1\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\boldsymbol{W}_1^\top\right) \\
&= \frac{\rho}{\left\|\boldsymbol{w}_2\boldsymbol{m}^\top\left(\mathbb{E}\left[y_q\boldsymbol{z}^\top\right] - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2\boldsymbol{m}^\top\right)^\top\right\|_2}\alpha\left(\boldsymbol{w}_2\boldsymbol{m}^\top\boldsymbol{m}\boldsymbol{w}_2^\top\right) \\
&= \frac{\rho\alpha\boldsymbol{w}_2\boldsymbol{w}_2^\top}{\|\boldsymbol{w}_2\|_2\left|\|\mathbb{E}[y_q\boldsymbol{z}]\|_2 - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2\right|} \quad \text{by } \boldsymbol{m} = \frac{\mathbb{E}[y_q\boldsymbol{z}^\top]}{\|\mathbb{E}[y_q\boldsymbol{z}]\|_2}.
\end{aligned}$$

Subsequently, we can approximate the SAM dynamics via:

$$\tau(\dot{\boldsymbol{w}}_2)_{\text{SAM}} = (\boldsymbol{I}+\boldsymbol{P})\tau(\dot{\boldsymbol{w}}_2)_{\text{GD}} = (\boldsymbol{I}+\boldsymbol{P})\boldsymbol{w}_2(\|\mathbb{E}[y_q\boldsymbol{z}]\| - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2).$$

To construct a solvable one-variable ODE, we consider $s = \boldsymbol{w}_2^\top\boldsymbol{w}_2$, which measures the evolution of weights via the norm, and let $\gamma = \|\mathbb{E}\left[y_q\boldsymbol{z}\right]\|_2$:

$$\begin{aligned}
\tau(\dot{s})_{\text{SAM}} = 2\boldsymbol{w}_2^\top\tau(\dot{\boldsymbol{w}}_2)_{\text{SAM}} &= 2\boldsymbol{w}_2^\top(\boldsymbol{I}+\boldsymbol{P})\boldsymbol{w}_2(\|\mathbb{E}[y_q\boldsymbol{z}]\|_2 - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2) \\
&= 2\boldsymbol{w}_2^\top\left(\boldsymbol{I} + \frac{\rho\alpha\boldsymbol{w}_2\boldsymbol{w}_2^\top}{\|\boldsymbol{w}_2\|_2\left|\gamma - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2\right|}\right)\boldsymbol{w}_2(\gamma - \alpha\boldsymbol{w}_2^\top\boldsymbol{w}_2) \\
&= 2s(\gamma-\alpha s) + 2\rho\alpha\left(\frac{s^2}{\sqrt{s}\left|\gamma-\alpha s\right|}\right)(\gamma-\alpha s) \\
&= 2s(\gamma-\alpha s) + 2\text{sign}(\gamma-\alpha s)\rho\alpha s^{3/2},
\end{aligned}$$

where by convention, $\text{sign}(x) = 0$ if $x = 0$, $= -1$ if $x < 0$, and $= 1$ if $x > 0$.

Suppose the initializations are sufficiently small in the sense that we start with $\gamma - \alpha s > 0$, where the dynamics $\tau(\dot{s})_{\text{SAM}}$ is "positive". Note that $s = \gamma/\alpha$ is a stable equilibrium. Therefore, s will increase from initialization until it reaches this equilibrium and stays here. It never reaches the "negative" part, so we can simplify to the following ODE for SAM:

$$\tau\frac{ds}{dt} = 2s(\gamma-\alpha s) + 2\rho\alpha s^{3/2}, \quad \text{where } \gamma - \alpha s > 0.$$

Lemma 9 shows the closed-form solution for the readers' reference:

$$\frac{1}{2\gamma}\ln\frac{s(t)}{s(0)} + \frac{1}{\alpha\Delta}\left(\frac{1}{r_-}\ln\frac{\sqrt{s(t)}-r_-}{\sqrt{s(0)}-r_-} - \frac{1}{r_+}\ln\frac{\sqrt{s(t)}-r_+}{\sqrt{s(0)}-r_+}\right) = \frac{t}{\tau}, \;\; r_\pm = \frac{\rho \pm \sqrt{\rho^2 + \frac{4\gamma}{\alpha}}}{2}.$$

Some algebra verifies that $\rho = 0$ ($r_+ = \sqrt{\gamma/\alpha},\ r_- = -r_+,\ \Delta = 2r_+$) yields:

$$\begin{aligned}
&\frac{1}{2\gamma}\ln\frac{s(t)}{s(0)} + \frac{1}{2\alpha r_+}\left(-\frac{1}{r_+}\ln\frac{\sqrt{s(t)}+r_+}{\sqrt{s(0)}+r_+} - \frac{1}{r_+}\ln\frac{\sqrt{s(t)}-r_+}{\sqrt{s(0)}-r_+}\right) \\
&= \frac{1}{2\gamma}\ln\frac{s(t)}{s(0)} - \frac{1}{2\alpha r_+^2}\left(\ln\frac{s(t)-r_+^2}{s(0)-r_+^2}\right) \\
&= \frac{1}{2\gamma}\ln\frac{s(t)}{s(0)} - \frac{1}{2\gamma}\left(\ln\frac{s(t)-\gamma/\alpha}{s(0)-\gamma/\alpha}\right) \\
&= \frac{1}{2\gamma}\ln\frac{s(t)(s(0)-\gamma/\alpha)}{s(0)(s(t)-\gamma/\alpha)} = \frac{t}{\tau}.
\end{aligned}$$

From the last equation, we can solve:

$$s(t) = \frac{(\gamma/\alpha)s(0)e^{2\gamma\frac{t}{\tau}}}{\gamma/\alpha + s(0)(e^{2\gamma\frac{t}{\tau}} - 1)} = \frac{\gamma e^{2\gamma\frac{t}{\tau}}}{\alpha(e^{2\gamma\frac{t}{\tau}} - 1) + \frac{\gamma}{s(0)}}.$$

This recovers the solution for GD dynamics in Zhang et al. (2025), which has meaningful empirical interpretations. □

Based on the ODE formula, we see that the SAM dynamics implements an additional boosting term $2\rho\alpha s^{3/2}$, which accelerates learning more strongly as s grows (the relative speed-up scales like $\sqrt{s}$), leading to a faster approach to the GD trajectory's operating region before saturation near $s = \gamma/\alpha$.

The variance parameter α plays a dual role: it *reduces* the steady-state level (since the GD carrying capacity is γ/α) while simultaneously *amplifying* the SAM gain. Intuitively, SAM is more effective at boosting learning when the features $\boldsymbol{z}$ are more visible (larger variance α).

While the implicit closed-form solution for SAM is less interpretable and thus not very interesting, we can simplify it given a small s. This yields an approximate ODE that **can extend to general covariances** and help quantify how long the model gets stuck around saddle points, as we show in the next section.

B.4 Escape Time from Saddles for General Covariance

We assume the **same time parameterization** for SAM and GD just as a sanity check, and then we compute the time for the model to move from small initialization to a larger early weight.

Theorem 8 (Escape Time)**.** *In the same setting as Theorem 7, except that we now allow general input covariance $\boldsymbol{\Sigma}$, under vanishing initializations $s_0 = s(0)$, we denote the time it takes for SAM and GD to reach a reasonably larger early weight s_* by t_{SAM}, t_{GD} respectively. Then SAM yields slightly faster "escape time":*

$$t_{GD} = \frac{\tau}{\|\boldsymbol{\Sigma}\|_F} \ln\left(\sqrt{\frac{s_*}{s_0}}\right)$$

$$t_{SAM} = \frac{\tau}{\|\boldsymbol{\Sigma}\|_F} \ln\left(\frac{\sqrt{s_*}(\|\boldsymbol{\Sigma}\|_F + \rho\alpha\sqrt{s_0})}{\sqrt{s_0}(\|\boldsymbol{\Sigma}\|_F + \rho\alpha\sqrt{s_*})}\right).$$

If $\|\boldsymbol{\Sigma}\|_F \gg \rho\alpha\sqrt{s_*}$, in particular,

$$\Delta t = t_{GD} - t_{SAM} = \frac{\tau\rho\alpha}{\|\boldsymbol{\Sigma}\|_F^2}(\sqrt{s_*} - \sqrt{s_0}).$$

Proof. Recall the following two ODEs from Section B.1:

1. **GD:** $\tau\frac{ds}{dt} = 2s(\gamma - \alpha s)$.
2. **SAM:** $\tau\frac{ds}{dt} = 2s(\gamma - \alpha s) + 2\rho\alpha s^{3/2}$.

Under sufficiently small initializations s and the fact that the weights have not moved much during early training, we drop *the least dominant term* and approximate the above ODEs as:

1. **GD:** $\tau\frac{ds}{dt} = 2\gamma s$.
2. **SAM:** $\tau\frac{ds}{dt} = 2\gamma s + 2\rho\alpha s^{3/2}$.

According to (Zhang et al., 2025; Atanasov et al., 2021)'s perspectives, (1) we can also view this as keeping *the most dominant term* in the gradient, which will yield the same ODE approximation via the new gradient:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}_2} = \nabla_{\boldsymbol{w}_2}\mathcal{L} = -2\boldsymbol{W}_1\left(\mathbb{E}\left[y_q\boldsymbol{z}^\top\right] - \underbrace{\boldsymbol{w}_2^\top\boldsymbol{W}_1}_{\text{two small weights}}\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\right)^\top \approx -2\boldsymbol{W}_1\mathbb{E}\left[y_q\boldsymbol{z}\right] + \text{small term}$$

$\implies$ the approximation: $\quad \tau \dot{\boldsymbol{w}}_2 = \boldsymbol{W}_1 \mathbb{E}\left[y_q \boldsymbol{z}\right].$

(2) Additionally, despite a general nonwhite covariance, for early training where the loss has not changed steeply, the same equation in (10) still holds

$$\boldsymbol{W}_1 = \boldsymbol{w}_2 \frac{\mathbb{E}[y_q \boldsymbol{z}^\top]}{\|\mathbb{E}[y_q \boldsymbol{z}]\|_2} \implies \tau \dot{\boldsymbol{w}}_2 = \boldsymbol{w}_2 \frac{\mathbb{E}[y_q \boldsymbol{z}^\top]}{\|\mathbb{E}[y_q \boldsymbol{z}]\|_2} \mathbb{E}\left[y_q \boldsymbol{z}\right] = \|\mathbb{E}[y_q \boldsymbol{z}]\|_2 \boldsymbol{w}_2 = \gamma \boldsymbol{w}_2.$$

Similarly, letting $s = \boldsymbol{w}_2^\top \boldsymbol{w}_2$ recovers exactly the ODE $\tau \frac{ds}{dt} = 2\gamma s$ for GD after dropping the least dominant term. And it follows that the corresponding SAM ODE is $\tau \frac{ds}{dt} = 2\gamma s + 2\rho\alpha s^{3/2}$.

The approximate ODE for GD, same as in the original paper, is a basic separable equation, which has the following solution:

$$s_{\text{GD}}(t) = s_{\text{GD}}(0) e^{2\gamma \frac{t}{\tau}}.$$

By Lemma 10, the SAM solution is:

$$s_{\text{SAM}}(t) = \left(\frac{\gamma \sqrt{s_{\text{SAM}}(0)} e^{\gamma \frac{t}{\tau}}}{\gamma + \rho\alpha \sqrt{s_{\text{SAM}}(0)} \left(1 - e^{\gamma \frac{t}{\tau}}\right)} \right)^2.$$

Note that $\rho = 0$ recovers the GD solution. The above solutions provide analytical and tractable evolutions for these weights. For the early phase of training, we can then compute the time required to reach some $s(t)$ i.e escape from the saddle.

Starting from the same small initializations where $s_0 := s_{\text{GD}}(0) = s_{\text{SAM}}(0)$, we want to reach some larger early point s_*. We let $t_{\text{GD}}, t_{\text{SAM}}$ denote the time required such that $s_{\text{GD}}(t_{\text{GD}}) = s_*$ and $s_{\text{SAM}}(t_{\text{SAM}}) = s_*$ respectively.

We can easily solve this by inverting the s_{GD}, s_{SAM} functions above and have the following:

$$t_{\text{GD}} = \frac{\tau}{\gamma} \ln\left(\sqrt{\frac{s_*}{s_0}}\right),$$

$$t_{\text{SAM}} = \frac{\tau}{\gamma} \ln\left(\frac{\sqrt{s_*}(\gamma + \rho\alpha\sqrt{s_0})}{\sqrt{s_0}(\gamma + \rho\alpha\sqrt{s_*})}\right).$$

Hence, we have:

$$\Delta t = t_{\text{GD}} - t_{\text{SAM}} = \frac{\tau}{\gamma} \ln\left(\frac{\gamma + \rho\alpha\sqrt{s_*}}{\gamma + \rho\alpha\sqrt{s_0}}\right) > 0 \quad \text{since} \quad s_* > s_0.$$

If $\gamma \gg \rho\alpha\sqrt{s_*}$ (e.g. $\alpha = 1 + (1+d)/N$, $\gamma = \sqrt{d}$ when the input covariance $\boldsymbol{\Sigma} = \boldsymbol{I}$ by Equation 50 from Zhang et al. (2025)), we can apply $\ln(1+x) \approx x$ for small x to obtain:

$$\Delta t = \frac{\tau}{\gamma}\left(\ln\left(1 + \frac{\rho\alpha\sqrt{s_*}}{\gamma}\right) - \ln\left(1 + \frac{\rho\alpha\sqrt{s_0}}{\gamma}\right)\right) \approx \frac{\tau}{\gamma}\left(\frac{\rho\alpha}{\gamma}\right)(\sqrt{s_*} - \sqrt{s_0}).$$

Hence, SAM allows for a slightly faster convergence in terms of the gradient flow, and $\gamma = \|\boldsymbol{\Sigma}\|_F^2$ by (Zhang et al., 2025) Equation 53 completes the proof.

□

Notably, since the escape time is inversely proportional to the Frobenius norm of data covariance, increasing the norm via amplifying any data can accelerate convergence for both optimizers and continually reduce Δt. This is also apparent from Theorem 7, as increasing $\gamma = \|\boldsymbol{\Sigma}\|_F^2$ will make the first term larger than the second term being subtracted and reduce its effect. In particular, this suggests that modifying data can be a viable path for GD to have SAM-like properties. However, **in practice, SAM should slow down convergence** in terms of both iterations and training time because the model spends time and steps correcting the adversarial moves. For vision models, Nguyen et al. (2024a) shows that this process enables uniform feature learning and reduces SB for better generalization. This contradiction could be due to that the model is too simple, and SAM often steps in a good direction instead of an adversarial one as intended.

With this being said, we only provide the discussion of $\texttt{ATTN}_M$ here, as it does not show relevant insights about SB.

B.5 Helper Lemmas

Lemma 9. *Consider the following ordinary differential equation:*

$$\tau \frac{ds}{dt} = 2s(\gamma - \alpha s) + 2\rho\alpha s^{3/2} \quad \textit{where } \gamma - \alpha s > 0.$$

Then let $\Delta = \sqrt{\rho^2 + \frac{4\gamma}{\alpha}}$*, the solution satisfies:*

$$\frac{1}{2\gamma}\ln\frac{s(t)}{s(0)} + \frac{1}{\alpha\Delta}\left(\frac{1}{r_-}\ln\frac{\sqrt{s(t)}-r_-}{\sqrt{s(0)}-r_-} - \frac{1}{r_+}\ln\frac{\sqrt{s(t)}-r_+}{\sqrt{s(0)}-r_+}\right) = \frac{t}{\tau}, \;\; r_\pm = \frac{\rho \pm \Delta}{2}.$$

Proof. Set $u(t) = \sqrt{s(t)}$. Skipping the argument t, we have $s = u^2$ and $\dot{s} = 2u\dot{u}$. Substituting into the ODE gives:

$$\tau(2u\dot{u}) = 2u^2(\gamma - \alpha u^2) + 2\rho\alpha u^3$$
$$\dot{u} = \frac{u}{\tau}\Big(\gamma + \rho\alpha u - \alpha u^2\Big).$$

This is separable:

$$\frac{du}{u(\gamma + \rho\alpha u - \alpha u^2)} = \frac{dt}{\tau}. \tag{11}$$

We perform partial fraction decomposition on the LHS. First,

$$\gamma + \rho\alpha u - \alpha u^2 = -\alpha(u - r_+)(u - r_-), \quad r_\pm = \frac{\rho \pm \sqrt{\rho^2 + 4\gamma/\alpha}}{2}.$$

We want to solve for A, B, C in the following equation:

$$\frac{1}{-\alpha u(u - r_+)(u - r_-)} = \frac{A}{u} + \frac{B}{u - r_+} + \frac{C}{u - r_-}$$
$$\implies 1 = -\alpha\left[A(u - r_+)(u - r_-) + Bu(u - r_-) + Cu(u - r_+)\right].$$

Let $u = 0$. This equation simplifies to:

$$1 = -\alpha A(-r_+)(-r_-) \quad \implies \quad A = \frac{1}{-\alpha r_+ r_-} = \frac{1}{\gamma}.$$

Similarly, with $u = r_+$ and $u = r_-$ respectively, we have:

$$1 = -\alpha B\, r_+(r_+ - r_-) \quad \implies \quad B = -\frac{1}{\alpha r_+(r_+ - r_-)}.$$

$$1 = -\alpha C r_-(r_- - r_+) = \alpha C r_-(r_+ - r_-) \quad \implies \quad C = \frac{1}{\alpha r_-(r_+ - r_-)}.$$

With the partial fraction decomposition, we integrate both sides of Equation 11:

$$\begin{aligned}
\int \frac{du}{u(\gamma + \rho\alpha u - \alpha u^2)} &= A\ln u + B\ln|u - r_+| + C\ln|u - r_-| \\
&= \frac{1}{\gamma}\ln u - \frac{1}{\alpha r_+(r_+ - r_-)}\ln|u - r_+| + \frac{1}{\alpha r_-(r_+ - r_-)}\ln|u - r_-| \\
&= \frac{1}{\gamma}\ln u + \frac{1}{\alpha\Delta}\left(\frac{1}{r_-}\ln|u - r_-| - \frac{1}{r_+}\ln|u - r_+|\right) \\
&= \frac{t}{\tau} + C,
\end{aligned}$$

where $\Delta = r_+ - r_- = \sqrt{\rho^2 + \frac{4\gamma}{\alpha}}$. Imposing the initial condition $u(0)$ gives

$$\frac{1}{\gamma}\ln\frac{u(t)}{u_0} + \frac{1}{\alpha\Delta}\left(\frac{1}{r_-}\ln\frac{u(t) - r_-}{u_0 - r_-} - \frac{1}{r_+}\ln\frac{u(t) - r_+}{u_0 - r_+}\right) = \frac{t}{\tau}.$$

Finally, substituting back $u(t) = \sqrt{s(t)}$ completes the proof. □

Lemma 10. *Consider the following ordinary differential equation:*

$$\tau \frac{ds}{dt} = 2\gamma s + 2\rho\alpha s^{3/2}, \quad \textit{where } \gamma - \alpha s > 0.$$

It has the following solution:

$$s(t) = \left(\frac{\gamma \sqrt{s(0)} e^{\gamma \frac{t}{\tau}}}{\gamma + \rho\alpha \sqrt{s(0)} \left(1 - e^{\gamma \frac{t}{\tau}}\right)} \right)^2.$$

Proof. Set $u(t) = \sqrt{s(t)}$. We have $s = u^2$ and $\dot{s} = 2u\dot{u}$. Substituting into the ODE gives:

$$\tau(2u\dot{u}) = 2\gamma u^2 + 2\rho\alpha u^3 \quad \Longrightarrow \quad \dot{u} = \frac{u}{\tau}\left(\gamma + \rho\alpha u\right).$$

This equation is separable:

$$\frac{du}{u(\gamma + \rho\alpha u)} = \frac{dt}{\tau}. \tag{12}$$

We perform partial fraction decomposition on the LHS and obtain,

$$\int \frac{1}{u(\gamma + \rho\alpha u)} du = \int \frac{1}{\gamma}\frac{1}{u} - \frac{\rho\alpha}{\gamma}\frac{1}{\gamma + \rho\alpha u} du \implies$$
$$\frac{1}{\gamma}\ln u - \frac{1}{\gamma}\ln(\gamma + \rho\alpha u) = \frac{1}{\gamma}\ln\left(\frac{u}{\gamma + \rho\alpha u}\right) = \frac{t}{\tau} + C.$$

Imposing the initial condition $u(0)$ gives

$$\ln\left(\frac{u}{\gamma + \rho\alpha u}\right) - \ln\left(\frac{u(0)}{\gamma + \rho\alpha u(0)}\right) = \gamma\frac{t}{\tau} \implies u(t) = \left(\frac{\gamma u(0) e^{\gamma \frac{t}{\tau}}}{\gamma + \rho\alpha u(0) \left(1 - e^{\gamma \frac{t}{\tau}}\right)} \right)^2.$$

Finally, substituting back $u(t) = \sqrt{s(t)}$ completes the proof. □

C Proofs of Theorems for $\texttt{ATTN}_S$

We now focus on proving Theorems 3 and 5 in the main text, showing that SAM reduces SB early in training.

C.1 Preliminary and Our Hessian Extension

We first recall and summarize some important properties and formulas from (Zhang et al., 2025) (with slightly different notations), in particular from Section E and Subsections C.1, C.2. Given the model $\texttt{ATTN}_S$, we have the following gradients for the i-th head:

$$\tau \dot{v}_i = \boldsymbol{k}_i^\top \mathbb{E}\left[\boldsymbol{\beta}\left(y_q - \hat{y}_q\right)\boldsymbol{x}_q^\top\right]\boldsymbol{q}_i = \boldsymbol{k}_i^\top \left(\boldsymbol{\Sigma}^2 - \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\boldsymbol{\Sigma}\right)\boldsymbol{q}_i, \tag{13}$$

$$\tau \dot{\boldsymbol{k}}_i = v_i \mathbb{E}\left[\boldsymbol{\beta}\left(y_q - \hat{y}_q\right)\boldsymbol{x}_q^\top\right]\boldsymbol{q}_i = v_i \left(\boldsymbol{\Sigma}^2 - \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\boldsymbol{\Sigma}\right)\boldsymbol{q}_i, \tag{14}$$

$$\tau \dot{\boldsymbol{q}}_i = v_i \mathbb{E}\left[\boldsymbol{x}_q\left(y_q - \hat{y}_q\right)\boldsymbol{\beta}^\top\right]\boldsymbol{k}_i = v_i \left(\boldsymbol{\Sigma}^2 - \boldsymbol{\Sigma}\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\right)\boldsymbol{k}_i. \tag{15}$$

Equivalently, for the squared loss, we have the following gradient:

$$\frac{\partial \mathcal{L}}{\partial v_i} = \nabla_{v_i}\mathcal{L} = -2\boldsymbol{k}_i^\top \left(\boldsymbol{\Sigma}^2 - \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\boldsymbol{\Sigma}\right)\boldsymbol{q}_i, \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{k}_i} = \nabla_{\boldsymbol{k}_i}\mathcal{L} = -2v_i \left(\boldsymbol{\Sigma}^2 - \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\boldsymbol{\Sigma}\right)\boldsymbol{q}_i, \tag{17}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_i} = \nabla_{\boldsymbol{q}_i}\mathcal{L} = -2v_i \left(\boldsymbol{\Sigma}^2 - \boldsymbol{\Sigma}\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\right)\boldsymbol{k}_i. \tag{18}$$

Here $\boldsymbol{\Sigma} = \sum_{i=1}^{d}\lambda_i \boldsymbol{e}_i\boldsymbol{e}_i^\top$ and $\hat{\boldsymbol{\Sigma}}$ denote the population covariance and sample covariance matrices, respectively. In particular, their Equations 30 and 31 show that:

$$\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right] = \boldsymbol{\Sigma}^2 + \frac{\boldsymbol{\Sigma} + \mathrm{Tr}(\boldsymbol{\Sigma})\boldsymbol{I}}{N}\boldsymbol{\Sigma} = \sum_{i=1}^{d} a_i \boldsymbol{e}_i\boldsymbol{e}_i^\top,$$

where the eigenvectors are the same as $\boldsymbol{\Sigma}$ and

$$a_i = \left(1 + \frac{1}{N}\right)\lambda_i^2 + \frac{\mathrm{Tr}(\boldsymbol{\Sigma})}{N}\lambda_i = \lambda_i^2\left(1 + \frac{1 + \mathrm{Tr}(\boldsymbol{\Sigma})/\lambda_i}{N}\right).$$

On top of the gradient computations, we perform the following Hessian computation for the i-th head weight $\boldsymbol{w}_i = \begin{bmatrix} v_i & \boldsymbol{k}_i^\top & \boldsymbol{q}_i^\top \end{bmatrix}^\top \in \mathbb{R}^{2d+1}$ ("with itself"). The Hessian has the 3×3 block structure

$$\boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{H}_{i,vv} & \boldsymbol{H}_{i,vk} & \boldsymbol{H}_{i,vq} \\ \boldsymbol{H}_{i,kv} & \boldsymbol{H}_{i,kk} & \boldsymbol{H}_{i,kq} \\ \boldsymbol{H}_{i,qv} & \boldsymbol{H}_{i,qk} & \boldsymbol{H}_{i,qq} \end{bmatrix}, \quad \text{where}$$

$$\boldsymbol{H}_{i,vv} = \frac{\partial}{\partial v_i}\left(\nabla_{v_i}\mathcal{L}\right) = 2\boldsymbol{k}_i^\top \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{k}_i\boldsymbol{q}_i^\top\boldsymbol{\Sigma}\boldsymbol{q}_i,$$

$$\boldsymbol{H}_{i,kv} = \boldsymbol{H}_{i,vk}^\top = \frac{\partial}{\partial \boldsymbol{k}_i}\left(\nabla_{v_i}\mathcal{L}\right) = -2\boldsymbol{\Sigma}^2\boldsymbol{q}_i + 2\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\boldsymbol{\Sigma}\boldsymbol{q}_i + 2v_i\boldsymbol{q}_i^\top\boldsymbol{\Sigma}\boldsymbol{q}_i\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{k}_i,$$

$$\boldsymbol{H}_{i,kk} = \frac{\partial}{\partial \boldsymbol{k}_i^\top}\left(\nabla_{\boldsymbol{k}_i}\mathcal{L}\right) = 2v_i^2\boldsymbol{q}_i^\top\boldsymbol{\Sigma}\boldsymbol{q}_i\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right],$$

$$\boldsymbol{H}_{i,kq} = \boldsymbol{H}_{i,qk}^\top = \frac{\partial}{\partial \boldsymbol{q}_i^\top}\left(\nabla_{\boldsymbol{k}_i}\mathcal{L}\right) = -2v_i\boldsymbol{\Sigma}^2 + 2v_i\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i'=1}^{H} v_{i'}\boldsymbol{k}_{i'}\boldsymbol{q}_{i'}^\top\boldsymbol{\Sigma} + 2v_i^2\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{k}_i\boldsymbol{q}_i^\top\boldsymbol{\Sigma},$$

$$\boldsymbol{H}_{i,qq} = \frac{\partial}{\partial \boldsymbol{q}_i^\top}\left(\nabla_{\boldsymbol{q}_i}\mathcal{L}\right) = 2v_i^2\boldsymbol{\Sigma}\boldsymbol{k}_i\boldsymbol{k}_i^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right],$$

$$\boldsymbol{H}_{i,qv} = \boldsymbol{H}_{i,vq}^\top = \frac{\partial}{\partial \boldsymbol{q}_i}\left(\nabla_{v_i}\mathcal{L}\right) = -2\boldsymbol{\Sigma}^2\boldsymbol{k}_i + 2\boldsymbol{\Sigma}\sum_{i'=1}^{H} v_{i'}\boldsymbol{q}_{i'}\boldsymbol{k}_{i'}^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{k}_i + 2v_i\boldsymbol{k}_i^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{k}_i\boldsymbol{\Sigma}\boldsymbol{q}_i.$$

In the above Hessian, we do not need to include other heads, as the heads become active for learning one at a time, and the others do not induce new gradient in SAM.

C.2 Learning the First Dominant Eigenvector: Proof of Theorem 3 Part 1

We permute and index the heads by the order of learning the eigenvectors. At the start of training, the model learns the most prominent eigenvector $\boldsymbol{e}_1$, and recall that by the ansatz, the weights are set as:

$$\boldsymbol{k}_1 = \boldsymbol{q}_1 = v_1\boldsymbol{e}_1; \quad \boldsymbol{k}_i = \boldsymbol{q}_i = \boldsymbol{0},\ v_i = 0 \quad \text{for other heads.}$$

The head indexed 1 first becomes active, and the other parallel heads are not affected during the training (zero dynamics).

We first have the following simplified GD dynamics from Equations 13, 14, 15:

$$\boldsymbol{\Sigma}^2 - \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\sum_{i=1}^{H} v_i\boldsymbol{k}_i\boldsymbol{q}_i^\top\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^2 - \sum_{i=1}^{d} a_i\boldsymbol{e}_i\boldsymbol{e}_i^\top v_1^3\boldsymbol{e}_1\boldsymbol{e}_1^\top\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^2 - \lambda_1 a_1\boldsymbol{e}_1\boldsymbol{e}_1^\top v_1^3$$

$$\tau\dot{v}_1 = v_1^2\boldsymbol{e}_1^\top\left(\boldsymbol{\Sigma}^2 - \lambda_1 a_1\boldsymbol{e}_1\boldsymbol{e}_1^\top v_1^3\right)\boldsymbol{e}_1 = \lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5, \quad \tau\dot{\boldsymbol{k}}_1 = \lambda_1^2 v_1^2\boldsymbol{e}_1 - \lambda_1 a_1 v_1^5\boldsymbol{e}_1,$$

$$\tau\dot{\boldsymbol{q}}_1 = \lambda_1^2 v_1^2\boldsymbol{e}_1 - \lambda_1 a_1 v_1^5\boldsymbol{e}_1, \quad v_i = 0,\ \boldsymbol{k}_i = 0,\ \boldsymbol{q}_i = 0, \quad i = 2, \ldots, H.$$

This significantly reduces the complexity of the dynamics. Note that we simply multiply by -2 to obtain the gradients. Now we want to introduce SAM and extend the computation to the Hessian:

$$\boldsymbol{H}_{1,vv} = 2\boldsymbol{k}_1^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{k}_1\boldsymbol{q}_1^\top\boldsymbol{\Sigma}\boldsymbol{q}_1 = 2v_1^4\boldsymbol{e}_1^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1\boldsymbol{e}_1^\top\boldsymbol{\Sigma}\boldsymbol{e}_1 = 2\lambda_1 a_1 v_1^4.$$

Similarly, we can compute

$$\boldsymbol{H}_{1,kv} = \boldsymbol{H}_{1,vk}^\top = -2\lambda_1^2 v_1\boldsymbol{e}_1 + 2v_1^4\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1\boldsymbol{e}_1^\top\boldsymbol{\Sigma}\boldsymbol{e}_1 + 2v_1^4\boldsymbol{e}_1^\top\boldsymbol{\Sigma}\boldsymbol{e}_1\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1 = (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1,$$

$$\boldsymbol{H}_{1,kk} = 2v_1^4\boldsymbol{e}_1^\top\boldsymbol{\Sigma}\boldsymbol{e}_1\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right] = 2\lambda_1 v_1^4\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right],$$

$$\boldsymbol{H}_{1,kq} = -2v_1\boldsymbol{\Sigma}^2 + 2v_1^4\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1\boldsymbol{e}_1^\top\boldsymbol{\Sigma} + 2v_1^4\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1\boldsymbol{e}_1^\top\boldsymbol{\Sigma} = -2v_1\boldsymbol{\Sigma}^2 + 4\lambda_1 a_1 v_1^4\boldsymbol{e}_1\boldsymbol{e}_1^\top,$$

$$\boldsymbol{H}_{1,qq} = 2v_1^4\boldsymbol{\Sigma}\boldsymbol{e}_1\boldsymbol{e}_1^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right] = 2\lambda_1 a_1 v_1^4\boldsymbol{e}_1\boldsymbol{e}_1^\top,$$

$$\boldsymbol{H}_{1,qv} = -2v_1\boldsymbol{\Sigma}^2\boldsymbol{e}_1 + 2v_1^4\boldsymbol{\Sigma}\boldsymbol{e}_1\boldsymbol{e}_1^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1 + 2v_1^4\boldsymbol{e}_1^\top\mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right]\boldsymbol{e}_1\boldsymbol{\Sigma}\boldsymbol{e}_1 = (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1.$$

Recall that the SAM and GD dynamics can be related via the following first-order approximation:

$$\tau(\dot{\boldsymbol{w}}_1)_{\text{SAM}} = \begin{bmatrix}\tau\dot{v}_1\\ \tau\dot{\boldsymbol{k}}_1\\ \tau\dot{\boldsymbol{q}}_1\end{bmatrix}_{\text{SAM}} = (\boldsymbol{I}_{2d+2} + \boldsymbol{P})\begin{bmatrix}\tau\dot{v}_1\\ \tau\dot{\boldsymbol{k}}_1\\ \tau\dot{\boldsymbol{q}}_1\end{bmatrix}_{\text{GD}}, \quad \boldsymbol{P} = \frac{\rho}{\|\nabla_{\boldsymbol{w}_1}\mathcal{L}\|_2}\boldsymbol{H}_1. \tag{19}$$

We plug in the relevant quantities:

$$(\boldsymbol{I}_{2d+2} + \boldsymbol{P})\begin{bmatrix}\tau\dot{v}_1\\ \tau\dot{\boldsymbol{k}}_1\\ \tau\dot{\boldsymbol{q}}_1\end{bmatrix}_{\text{GD}} = \underbrace{\begin{bmatrix}\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5\\ (\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5)\boldsymbol{e}_1\\ (\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5)\boldsymbol{e}_1\end{bmatrix}}_{\text{GD dynamics}} + \underbrace{\boldsymbol{P}\begin{bmatrix}\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5\\ (\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5)\boldsymbol{e}_1\\ (\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5)\boldsymbol{e}_1\end{bmatrix}}_{\text{SAM correction} =: \boldsymbol{Q}}$$

It remains to compute the perturbation matrix. First note $\|\nabla_{\boldsymbol{w}_1}\mathcal{L}\|_2 = \sqrt{\|\nabla_{v_1}\mathcal{L}\|_2 + \|\nabla_{\boldsymbol{k}_1}\mathcal{L}\|_2 + \|\nabla_{\boldsymbol{q}_1}\mathcal{L}\|_2} = 2\sqrt{3}|\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5|$ by Equations 16, 17, 18. Furthermore, due to sufficiently small initializations, $|\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5| = \lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5$ and the SAM correction term, according to 19, becomes:

$$\begin{aligned}
\boldsymbol{Q} &= \frac{\rho}{2\sqrt{3}} \begin{bmatrix} 2\lambda_1 a_1 v_1^4 & (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1^\top & (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1^\top \\ (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1 & 2\lambda_1 v_1^4 \mathbb{E}[\hat{\boldsymbol{\Sigma}}^2] & -2v_1\boldsymbol{\Sigma}^2 + 4\lambda_1 a_1 v_1^4 \boldsymbol{e}_1\boldsymbol{e}_1^\top \\ (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1 & -2v_1\boldsymbol{\Sigma}^2 + 4\lambda_1 a_1 v_1^4 \boldsymbol{e}_1\boldsymbol{e}_1^\top & 2\lambda_1 a_1 v_1^4 \boldsymbol{e}_1\boldsymbol{e}_1^\top \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{e}_1 \\ \boldsymbol{e}_1 \end{bmatrix} \\
&= \frac{\rho}{2\sqrt{3}} \begin{bmatrix} 2\lambda_1 a_1 v_1^4 + (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4) + (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4) \\ (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1 + 2\lambda_1 a_1 v_1^4 \boldsymbol{e}_1 - 2\lambda_1^2 v_1 \boldsymbol{e}_1 + 4\lambda_1 a_1 v_1^4 \boldsymbol{e}_1 \\ (-2\lambda_1^2 v_1 + 4\lambda_1 a_1 v_1^4)\boldsymbol{e}_1 - 2\lambda_1^2 v_1 \boldsymbol{e}_1 + 4\lambda_1 a_1 v_1^4 \boldsymbol{e}_1 + 2\lambda_1 a_1 v_1^4 \boldsymbol{e}_1 \end{bmatrix} \\
&= \frac{\rho}{2\sqrt{3}} \begin{bmatrix} 10\lambda_1 a_1 v_1^4 - 4\lambda_1^2 v_1 \\ (10\lambda_1 a_1 v_1^4 - 4\lambda_1^2 v_1)\boldsymbol{e}_1 \\ (10\lambda_1 a_1 v_1^4 - 4\lambda_1^2 v_1)\boldsymbol{e}_1 \end{bmatrix}.
\end{aligned}$$

Finally, here we drop the subscript and have the following SAM dynamics:

$$\begin{aligned}
\tau\dot{v}_1 &= \lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5 + \frac{\rho}{\sqrt{3}}\left(5\lambda_1 a_1 v_1^4 - 2\lambda_1^2 v_1\right), \\
\tau\dot{\boldsymbol{k}}_1 &= \left(\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5 + \frac{\rho}{\sqrt{3}}\left(5\lambda_1 a_1 v_1^4 - 2\lambda_1^2 v_1\right)\right)\boldsymbol{e}_1, \\
\tau\dot{\boldsymbol{q}}_1 &= \left(\lambda_1^2 v_1^2 - \lambda_1 a_1 v_1^5 + \frac{\rho}{\sqrt{3}}\left(5\lambda_1 a_1 v_1^4 - 2\lambda_1^2 v_1\right)\right)\boldsymbol{e}_1.
\end{aligned}$$

Similar to Section B.4, under small initializations, we can drop the lower order terms and obtain the following ODE's for SAM and GD:

GD Dynamics:

$$\tau\dot{v}_1 = \lambda_1^2 v_1^2, \qquad \tau\dot{\boldsymbol{k}}_1 = \lambda_1^2 v_1^2 \boldsymbol{e}_1, \qquad \tau\dot{\boldsymbol{q}}_1 = \lambda_1^2 v_1^2 \boldsymbol{e}_1.$$

SAM Dynamics:

$$\tau\dot{v}_1 = \lambda_1^2 v_1^2 - \frac{2\rho}{\sqrt{3}}\lambda_1^2 v_1, \qquad \tau\dot{\boldsymbol{k}}_1 = \left(\lambda_1^2 v_1^2 - \frac{2\rho}{\sqrt{3}}\lambda_1^2 v_1\right)\boldsymbol{e}_1, \qquad \tau\dot{\boldsymbol{q}}_1 = \left(\lambda_1^2 v_1^2 - \frac{2\rho}{\sqrt{3}}\lambda_1^2 v_1\right)\boldsymbol{e}_1.$$

We note that we can reduce them to one scalar-variable ODE:

1. **GD**: $\tau\dot{v}_1 = \lambda_1^2 v_1^2$,
2. **SAM**: $\tau\dot{v}_1 = \lambda_1^2 v_1^2 - \frac{2\rho}{\sqrt{3}}\lambda_1^2 v_1$.

C.3 Saddle-to-Saddle Learning: Proof of Theorem 3 Part 2

It turns out that when the model is at the $(m+1)$-th plateau for $m \geq 1$, namely the model is learning the $(m+1)$-th feature, the learning dynamics can be captured by the same ODEs. Recall from the ansatz that:

$$\boldsymbol{k}_i = \boldsymbol{q}_i = v_i \boldsymbol{e}_i = \left(\lambda_i + \frac{\lambda_i + \mathrm{Tr}(\boldsymbol{\Sigma})}{N}\right)^{-1/3} \boldsymbol{e}_i,\ 1 \leq i \leq m, \quad \boldsymbol{k}_{m+1} = \boldsymbol{q}_{m+1} = v_{m+1}(t)\boldsymbol{e}_{m+1}$$

$$\boldsymbol{k}_i = \boldsymbol{q}_i = \boldsymbol{0}, v_i = 0,\ m+2 \leq i \leq H \text{ (the other heads)}$$

We first have the following simplified GD dynamics for v_{m+1} from Equation 13:

$$\begin{aligned}
\tau\dot{v}_{m+1} &= \boldsymbol{k}_{m+1}^\top \left(\boldsymbol{\Sigma}^2 - \mathbb{E}\left[\hat{\boldsymbol{\Sigma}}^2\right] \sum_{i'=1}^{H} v_{i'} \boldsymbol{k}_{i'} \boldsymbol{q}_{i'}^\top \boldsymbol{\Sigma}\right) \boldsymbol{q}_{m+1} \\
&= \lambda_{m+1}^2 v_{m+1}^2 - v_{m+1}^2 \boldsymbol{e}_{m+1}^\top \sum_{i=1}^{d} a_i \boldsymbol{e}_i \boldsymbol{e}_i^\top \sum_{i'=1}^{H} v_{i'}^3 \boldsymbol{e}_{i'} \boldsymbol{e}_{i'}^\top \boldsymbol{e}_{m+1}
\end{aligned}$$

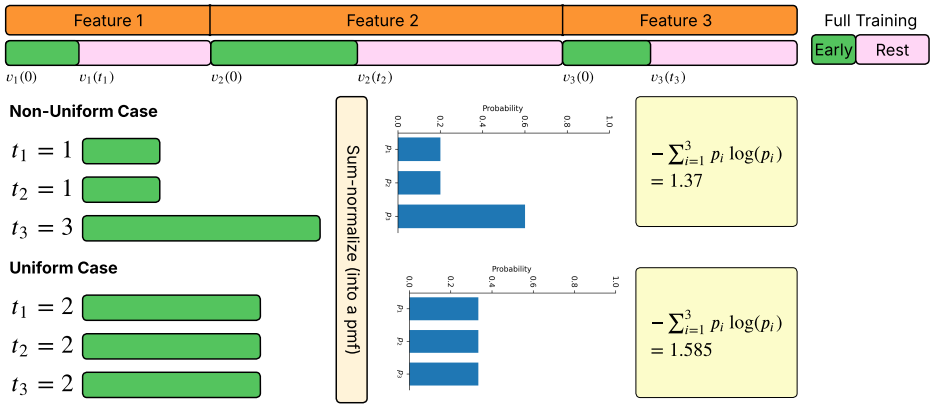

Figure 8: Toy example of Shannon entropy in quantifying uniformity in feature learning. We consider the extreme case of only 3 features. Following the $\texttt{ATTN}_S$ setting, the training can be separated by features. For each feature, there exists an early training phase (in green), and t_1, t_2, t_3 mark the times spent in this regime. More uniform learning is achieved when these values are closer to each other (bottom case), and after sum-normalization, this naturally corresponds to a distribution that resembles the uniform distribution (e.g. $(1/3, 1/3, 1/3)$ over three objects) and hence has higher entropy

$$= \lambda_{m+1}^2 v_{m+1}^2 - \lambda_{m+1} a_{m+1} v_{m+1}^5,$$

where the last step follows from orthonormal eigenvectors. This is the same dynamics as v_1 (just a different index), and the same applies for $\boldsymbol{k}_{m+1}$ and $\boldsymbol{q}_{m+1}$.

Calculating the SAM perturbation $\boldsymbol{P}$ further requires the Hessian $\boldsymbol{H}_{m+1}$, which follows the same process as computing $\boldsymbol{H}_1$. Hence, the perturbation part is also the same up to replacing the index.

C.4 Learning Entropy: Proof of Theorem 5

We first refer to Figure 8 to provide a toy instantiation of how the entropy of the normalized time sequence captures uniformity in feature learning. Note that after sum-normalization, the discrete time sequence is converted into a probability mass distribution, and more uniform learning (that is more even time spent on the features) naturally becomes "close" to a uniform distribution. We quantify the notion of closeness via Shannon entropy, for which the uniform distribution is the largest.

Now consider the following two functions:

$$I_1(u) := \int_\varepsilon^u \frac{1}{v^2}\, dv, \quad I_2(u) := \int_\varepsilon^u \frac{1}{v^2 - \hat{\rho} v}\, dv, \quad \hat{\rho} = \frac{2\rho}{\sqrt{3}}.$$

Then letting $u_i = c\lambda_i^{-1/3}$, we have:

$$t_i^{\text{GD}} = \frac{\tau}{\lambda_i^2} I_1(u_i) = \tau \left(\frac{u_i}{c}\right)^6 I_1(u_i), \quad t_i^{\text{SAM}} = \frac{\tau}{\lambda_i^2} I_2(u_i) = \tau \left(\frac{u_i}{c}\right)^6 I_2(u_i). \tag{20}$$

Step 1: Note that t_i^{GD} and t_i^{SAM} are both strictly increasing in u_i. This is straightforward as $I_1(u)$ and $I_2(u)$ are both increasing in u (when $\varepsilon > \hat{\rho}$).

Step 2: Show that t_i^{SAM} is a decreasing "reweighting" of t_i^{GD}. More precisely, we start by defining the ratio:

$$w(u) := \frac{I_2(u)}{I_1(u)} = \frac{t^{\text{SAM}}(u)}{t^{\text{GD}}(u)} \quad \text{if we treat them as continuous.}$$

We first note that:

$$w(u) = \frac{\int_\varepsilon^u \frac{1}{v^2 - \hat{\rho} v}\, dv}{\int_\varepsilon^u \frac{1}{v^2}\, dv} = \frac{\int_\varepsilon^u \phi(v) \frac{1}{v^2}\, dv}{\int_\varepsilon^u \frac{1}{v^2}\, dv}, \quad \text{where } \phi(v) = \frac{v}{v - \hat{\rho}}.$$

Note that $\phi(v)$ is strictly decreasing on $[\varepsilon, \infty)$ i.e. $\phi'(v) = \frac{-\hat{\rho}}{(v-\hat{\rho})^2} < 0$. This implies that:

$$w(u) = \frac{I_2(u)}{I_1(u)} > \phi(u) \frac{\int_\varepsilon^u \frac{1}{v^2}\, dv}{\int_\varepsilon^u \frac{1}{v^2}\, dv} = \phi(u) = \frac{u}{u-\hat{\rho}} = \frac{I_2'(u)}{I_1'(u)}. \tag{21}$$

We now differentiate and obtain:

$$\begin{aligned}\operatorname{sign}(w'(u)) = \operatorname{sign}\left(\frac{I_1(u)I_2'(u) - I_1'(u)I_2(u)}{I_1(u)^2}\right) &= \operatorname{sign}\left(I_1(u)I_2'(u) - I_1'(u)I_2(u)\right) \\ &= \operatorname{sign}\left(\frac{I_2'(u)}{I_1'(u)} I_1(u) - I_2(u)\right) < 0,\end{aligned}$$

where the last inequality directly follows from Equation 21. This shows the claim.

Step 3: It follows from Lemma 12 when we let:

The non-increasing sequence $A =$ the sequence $\dfrac{t_i^{\mathrm{GD}}}{\sum_j t_j^{\mathrm{GD}}}$ $(i = 1, \ldots, M)$ reversed (decreasing by Step 1);

The non-increasing sequence $B =$ the sequence $\dfrac{t_i^{\mathrm{SAM}}}{\sum_j t_j^{\mathrm{SAM}}}$ $(i = 1, \ldots, M)$ reversed (decreasing by Step 1);

The non-decreasing sequence $w =$ the sequence $\dfrac{t_i^{\mathrm{SAM}}}{t_i^{\mathrm{GD}}}$ $(i = 1, \ldots, M)$ reversed (increasing by Step 2),

we have that the normalized GD time sequence majorizes the normalized SAM time sequence. By the strict Schur-concavity of Shannon entropy (Lemma 14),

$$\textbf{Entropy}\left(\left\{t_i^{\mathrm{SAM}}\right\}_{i=1}^M\right) > \textbf{Entropy}\left(\left\{t_i^{\mathrm{GD}}\right\}_{i=1}^M\right).$$

This completes the proof of reduced simplicit bias.

C.5 Majorization Theory: Helper Lemmas

In mathematics, majorization theory is a natural branch for studying spread because it gives a principled way to say when one set of numbers is more uneven (more "peaked") than another—without committing to a single statistic like variance, which can be unreliable at times. We provide the following context for majorization theory:

Definition 11. (Restatement of Vector Majorization, Definition A.1, Chapter 1Marshall et al. (1979)) Let $x, y \in \mathbb{R}^n$. Let $x^\downarrow$ and $y^\downarrow$ denote the vectors obtained by sorting the coordinates of x and y in *nonincreasing* order:

$$x_1^\downarrow \geq x_2^\downarrow \geq \cdots \geq x_n^\downarrow, \qquad y_1^\downarrow \geq y_2^\downarrow \geq \cdots \geq y_n^\downarrow.$$

We say that x *majorizes* y, and write $x \succ y$, if

$$\sum_{i=1}^k x_i^\downarrow \;\geq\; \sum_{i=1}^k y_i^\downarrow \quad \text{for all } k = 1, 2, \ldots, n-1, \quad \text{and} \quad \sum_{i=1}^n x_i^\downarrow \;=\; \sum_{i=1}^n y_i^\downarrow.$$

Lemma 12 (Increasing-weight renormalization is majorized)**.** *Let $A = (A_1, \ldots, A_n)$ be a probability vector with*

$$A_1 \geq A_2 \geq \cdots \geq A_n > 0, \quad \sum_{i=1}^n A_i = 1.$$

Let $w = (w_1, \ldots, w_n)$ be a nondecreasing sequence of positive weights $0 < w_1 \leq w_2 \leq \cdots \leq w_n$, and define the renormalized reweighting $B = (B_1, \ldots, B_n)$ by

$$B_i := \frac{A_i w_i}{\sum_{j=1}^n A_j w_j}, \qquad i = 1, \ldots, n.$$

Assume that $B_1 \geq B_2 \geq \cdots \geq B_n$. Then A majorizes B ($A \succ B$), i.e.

$$\sum_{i=1}^k A_i \;\geq\; \sum_{i=1}^k B_i \quad \textit{for } k = 1, \ldots, n-1, \qquad \textit{and} \qquad \sum_{i=1}^n A_i = \sum_{i=1}^n B_i = 1.$$

Proof. Define the following partial sums and weighted average:

$$\alpha_k := \sum_{i=1}^{k} A_i, \quad \beta_k := \sum_{i=1}^{k} B_i = \frac{\sum_{i=1}^{k} A_i w_i}{\sum_{j=1}^{n} A_j w_j}, \quad m_k := \frac{\sum_{i=1}^{k} A_i w_i}{\sum_{i=1}^{k} A_i} = \frac{\sum_{i=1}^{k} A_i w_i}{\alpha_k}.$$

Since $(w_i)_{i=1}^n$ is increasing, we have $m_k \in [w_1, w_k]$, hence $m_k \le w_k \le w_{k+1}$, and

$$m_{k+1} = \frac{\alpha_k m_k + A_{k+1} w_{k+1}}{\alpha_k + A_{k+1}} \ge \frac{\alpha_k m_k + A_{k+1} m_k}{\alpha_k + A_{k+1}} = m_k.$$

Therefore $(m_k)_{k=1}^n$ is nondecreasing in k. In particular, $m_k \le m_n$, where $m_n = \sum_{j=1}^{n} A_j w_j$. Consequently,

$$\beta_k = \frac{\sum_{i=1}^{k} A_i w_i}{\sum_{j=1}^{n} A_j w_j} = \frac{\alpha_k m_k}{m_n} \le \frac{\alpha_k m_n}{m_n} = \alpha_k, \quad k = 1, \dots, n.$$

Also $\beta_n = \alpha_n = 1$. These are exactly the majorization inequalities:

$$\sum_{i=1}^{k} A_i \ \ge \ \sum_{i=1}^{k} B_i \quad (k = 1, \dots, n-1), \qquad \sum_{i=1}^{n} A_i \ = \ \sum_{i=1}^{n} B_i \ = \ 1.$$

Hence $A \succ B$ by Definition 11. □

Definition 13. (Restatement of Strict Schur-Concavity, Definition A.1, Chapter 3 Marshall et al. (1979)) Let $D \subseteq \mathbb{R}^n$ be a set that is closed under permutations of coordinates (i.e., if $x \in D$ then $Px \in D$ for any permutation matrix P). A function $f : D \to \mathbb{R}$ is called *strictly Schur-concave* if for all $x, y \in D$,

$$x \succ y \quad \Longrightarrow \quad f(x) \le f(y),$$

where $\succeq$ denotes (vector) majorization, and equality holds only when x is a permutation of y.

Lemma 14. *(Example D.1, Chapter 3 Marshall et al. (1979)) Shannon entropy is strictly Schur-concave.*

At the end of the proof, we provide a summary of the theoretical results in Table 1.

Table 1: Summary of theoretical results.

Model	Focus		Theory
$\texttt{ATTN}_M$	Training dynamics	ODE	Theorem 7: closed-form solution, convergence boosting term
		Escape time	Theorem 8: SAM escapes from saddles slightly faster
	Data upsampling	Any data	GD becomes similar to SAM if $\|\mathbf{\Sigma}\|_F \uparrow$
		"Less-rep." data	N/A
$\texttt{ATTN}_S$	Training dynamics	ODE	Theorem 3: SAM has an additional term that changes dynamics
		Escape time	Theorem 5: SAM reduces simplicity bias
	Data upsampling	Any data	N/A
		"Less-rep." data	Helps achieve uniform learning speed across features

D Implementation Details

D.1 Pseudocode

The pseudocode for our data-centric approach (§5) is given in Algorithm 1. A lightweight proxy model is briefly finetuned to extract per-example loss trajectories, which are clustered to identify less-represented examples, i.e., those whose underlying features the model has not yet had sufficient signal to learn. Such examples are then amplified via duplication or LLM-based rephrasing, prior to standard finetuning of the target model.

Algorithm 1 Proposed Trajectory-Based Data Upsampling

Require: Training dataset $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{N}$, proxy model M_0, number of checkpoints K, upsampling multiplier m (default: 1)

Output: Augmented training dataset $\widetilde{\mathcal{D}}$

▷ *Phase-1: Proxy trajectory estimation*
Finetune M_0 on $\mathcal{D}$ to save checkpoints $\{M^{(1)}, \ldots, M^{(K)}\}$ at evenly spaced intervals
for $i = 1$ **to** N **do**
    $\mathbf{t}_i \leftarrow \left[\mathcal{L}(M^{(1)}; x_i, y_i),\ \ldots,\ \mathcal{L}(M^{(K)}; x_i, y_i)\right]$ ▷ *loss trajectory*
end for
▷ *Phase-2: Cluster by feature representation*
Apply k-means (k=2) to $\{\mathbf{t}_i\}_{i=1}^{N}$, obtaining:
    $\mathcal{C}_{\text{well}}$: well-represented examples ▷ *low, rapidly decreasing loss*
    $\mathcal{C}_{\text{less}}$: less-represented examples ▷ *persistently elevated loss*
▷ *Phase-3: Amplify less-represented examples*
$\widetilde{\mathcal{D}} \leftarrow \mathcal{D}$
for each $(x_i, y_i) \in \mathcal{C}_{\text{less}}$ **do**
    Add m copies of (x_i, y_i) to $\widetilde{\mathcal{D}}$ ▷ *duplication*
    [Optional] Generate synthetic rephrase (x_i', y_i) via LLADA; add to $\widetilde{\mathcal{D}}$ ▷ *see Appendix G*
end for
▷ *Phase-4: Train the target model*
Finetune target model on $\widetilde{\mathcal{D}}$ with standard optimizer
return $\widetilde{\mathcal{D}}$

D.2 Hyperparameters

We finetune decoder-only LLMs using LoRA adapters, with task-specific hyperparameters fixed across model families within each task. As noted in §6, we use AdamW as the default optimizer, a maximum sequence length of 512, and random seed 0; model-specific LoRA target modules are reported in Table 2 and task-specific hyperparameters are in Table 3. All models are trained on 4× NVIDIA RTX A6000 GPUs.

For evaluation, we use vLLM-based inference and run experiments on 4× NVIDIA RTX A6000 or 4× NVIDIA H100 GPUs. Across all domains, we report the performance of the final-epoch checkpoint with the highest average accuracy across all benchmarks.

Table 2: **Model-specific LoRA target modules.** For all models except Phi2-2.7B, LoRA adapters are applied only to the self-attention projections; for Phi2-2.7B, adapters are also applied to the feed-forward layers. Pythia-70M is only used as the lightweight proxy for loss trajectory clustering.

Model	LoRA target modules
Phi2-2.7B	`q_proj, k_proj, v_proj, fc1, fc2`
Llama3.2-1B	`q_proj, k_proj, v_proj, o_proj`
Qwen3-0.6B-Base	`q_proj, k_proj, v_proj, o_proj`
Gemma3-1B-PT	`q_proj, k_proj, v_proj, o_proj`
Pythia-70M	`query_key_value, dense, dense_h_to_4h, dense_4h_to_h`

Table 3: **Task-specific hyperparameters.** Hyperparameters are fixed across all model families within each task. AdamW is used as the default optimizer across all tasks; Muon is additionally used for math to confirm optimizer-agnostic behavior. Pythia-70M trajectory length refers to the number of proxy checkpoints used for loss trajectory clustering.

	Math	Code	Instruction-following
Effective batch size	128	64	64
Max sequence length	512	512	512
Optimizer	AdamW, Muon	AdamW	AdamW
Learning rate	2×10^{-5}, 1×10^{-4}	2×10^{-5}	2×10^{-4}
Weight decay	0.0	0.0	0.0
LR scheduler	Cosine	Cosine	Cosine
Warmup ratio	0.03	0.03	0.05
LoRA rank: r	256	64	64
LoRA alpha: α	512	128	64
LoRA dropout	0.05	0.05	0.05
Save steps	256	64	64
Num. train epochs	3	4	3
Pythia trajectory length	5	9	12

E Detailed Experimental Results

We explicitly note that code evaluation (in Table 6) suffers from higher variance across seeds due to the brittle, binary nature of execution unit tests (e.g., minor indentation or syntax variations causing test failure), yet the average gain ($+4.88\%$ overall, $+9.75\%$ on HE, $+10.05\%$ on HE+) substantially exceeds the standard deviation.

Table 4: **Comparison of upsampling and synthetic augmentation across models and math benchmarks.** Each model was finetuned on MathInstruct using AdamW under three settings: original data (FT), exact duplication (FT+UP), and LLaDA-based synthetic augmentation (FT+SN), and evaluated zero-shot using greedy decoding. For each model, we report the last-epoch checkpoint with the highest average accuracy. Pretrained model performance (PT) is included for reference. For each model and row, accuracies within 1.0 percentage point of the best are indicated in **bold**, based on values before rounding.

	Qwen3-0.6B-Base				Llama3.2-1B				Gemma3-1B-PT				Phi2-2.7B			
Dataset	PT	FT	FT+UP	FT+SN	PT	FT	FT+UP	FT+SN	PT	FT	FT+UP	FT+SN	PT	FT	FT+UP	FT+SN
GSM8K	6.6	**58.3**	**59.2**	**58.9**	3.0	18.5	18.5	**21.9**	1.9	10.2	10.8	**12.7**	56.2	72.0	**73.7**	72.3
MATH	5.9	**35.8**	**35.8**	**35.7**	3.6	16.0	**17.0**	**17.6**	2.3	11.4	**12.4**	**12.8**	16.8	**34.3**	**35.3**	**35.1**
NumGLUE	9.5	60.1	60.7	**62.3**	16.4	34.1	**36.1**	**35.7**	7.0	30.7	**34.3**	**34.1**	34.7	62.2	**63.4**	**62.6**
SVAMP	6.8	76.3	**78.2**	77.0	8.5	**42.2**	37.9	**42.0**	3.3	31.3	32.3	**33.6**	67.8	**79.7**	**79.6**	**80.4**
DeepMind	7.1	**64.2**	61.2	**64.4**	6.0	33.5	35.5	**37.7**	5.0	26.3	28.4	**29.7**	33.9	56.8	57.4	**60.5**
MMLU-Math	**24.7**	22.8	**25.1**	22.3	16.2	24.2	**26.8**	23.2	12.0	**17.9**	**17.6**	15.6	12.4	**35.8**	**36.3**	**35.9**
Avg.	10.1	**52.92**	**53.36**	**53.43**	9.0	28.08	28.63	**29.69**	5.3	21.31	**22.62**	**23.08**	37.0	**56.81**	**57.62**	**57.80**

Table 5: **Comparison of upsampling with curriculum learning and OHEM across math benchmarks.** Llama3.2-1B was finetuned on MathInstruct with AdamW under three settings: FT (no upsampling), FT+UP (upsampling); CL (curriculum learning), CL+UP (curriculum learning with upsampling); OHEM (online hard example mining), OHEM+UP (OHEM with upsampling), and evaluated zero-shot with greedy decoding. For each model, we report the last-epoch checkpoint with the highest average accuracy. The highest performance *within each setting* (within 1.0%) is indicated in **bold**. For OHEM, at each optimizer step, we sample a pool of $2{,}048$ examples, score each using no-gradient forward pass, and select 128 examples via temperature-scaled softmax sampling (w/out replacement) biased toward higher-loss examples. Upsampling uses Pythia-70M as the proxy.

		Standard		Curriculum		OHEM	
Dataset	#Inst.	FT	FT+UP	CL	CL+UP	OHEM	OHEM+UP
GSM8K	$1,319$	22.7	**25.6**	30.1	**34.4**	24.3	**28.4**
MATH	$5,000$	14.6	**17.7**	**16.0**	**15.7**	**12.9**	**12.8**
NumGLUE	$1,042$	31.5	**36.1**	41.8	**46.2**	37.3	**44.5**
SVAMP	$1,000$	**41.7**	40.2	48.8	**50.7**	39.8	**45.7**
DeepMind	$1,000$	31.4	**34.5**	**36.9**	33.6	28.9	**32.6**
MMLU-Math	974	**27.2**	25.8	**13.9**	**14.1**	**23.2**	21.6
Avg.	$10,335$	28.2	**30.0**	31.3	**32.4**	27.7	**30.9**

E.1 Effect of Model Scale and Pretraining

Our results in Table 4 demonstrate that the efficacy of targeted upsampling is determined by the interplay between a model's intrinsic capacity and its pretraining exposure, rather than parameter scale alone. Modern 1B models, such as Gemma3-1B and Llama3.2-1B, exhibit highly expressive architectures that easily get trapped by superficial prompt heuristics during standard SFT; mitigating simplicity bias (FT+UP) successfully forces these models to build robust reasoning features, unlocking massive latent headroom. Conversely, models at the scale extremes exhibit different dynamics: Qwen3-0.6B benefits less as standard SFT primarily acts as a formatting trigger for its STEM-saturated pretraining, with absolute performance bounded by its physical 0.6B capacity. Phi2-2.7B registers a steady improvement, but its synthetic textbook pretraining on high-complexity reasoning tasks inherently resisted simplicity bias from the outset, narrowing the margin for downstream SFT

Table 6: **Comparison of upsampling across models and code benchmarks.** Each model was finetuned on the Python portion Magicoder-OSS-Instruct-75K (filtered to solutions between 50–500 tokens) using AdamW under two settings: without upsampling (FT) and with upsampling (FT+UP), and evaluated zero-shot with greedy decoding. For each model, we report the last-epoch checkpoint with the highest average accuracy. The highest performance (within 1.0%) is indicated in **bold**. Upsampling uses Pythia-70M as the proxy. Consistent with the finetuning setup, we only evaluate on instances with prompt length under 512 tokens.

			Qwen3-0.6B-Base		Llama3.2-1B	
Dataset	Ver.	#Inst.	FT	FT+UP	FT	FT+UP
HumanEval	HE	164	17.10 ± 1.69	$\mathbf{26.83 \pm 6.04}$	20.1	**22.6**
	HE+	164	14.62 ± 0.02	$\mathbf{24.67 \pm 4.71}$	**17.7**	**18.3**
MBPP	MBPP	971	$\mathbf{41.11 \pm 3.68}$	$\mathbf{41.21 \pm 3.26}$	**29.0**	**29.4**
	MBPP+	377	$\mathbf{44.94 \pm 3.59}$	$\mathbf{44.43 \pm 3.57}$	32.4	**33.4**
Avg.		$1,676$	29.42 ± 2.28	$\mathbf{34.30 \pm 1.00}$	24.8	**25.9**

Table 7: **MT-Bench scores by category.** Each model was finetuned on Alpaca-cleaned using AdamW under two settings: without upsampling (FT) and with upsampling (FT+UP), and evaluated on MT-Bench using GPT-4 Turbo. Math and coding categories are excluded to avoid overlap with our other benchmarks. The highest score (within a tolerance of 0.05) is indicated in **bold**.

	Llama3.2-1B		Qwen3-0.6B-Base	
Category	FT	FT+UP	FT	FT+UP
Writing	**5.15**	**5.15**	4.00	**4.40**
Roleplay	4.45	**5.10**	4.45	**4.65**
Reasoning	3.55	**3.70**	3.60	**3.70**
Extraction	**3.10**	2.60	4.25	**4.30**
STEM	4.00	**4.05**	5.05	**5.30**
Humanities	4.50	**5.20**	**5.30**	5.15
Overall	4.13	**4.30**	4.44	**4.58**

data interventions. Finally, Llama2-7B demonstrates that even in larger, overparameterized models that lack dense mathematical pretraining anchors, reducing simplicity bias remains highly effective in yielding consistent generalization improvements.

E.2 Confound Analysis

Our theoretical analysis predicts that examples with persistently high early-training loss contain less-represented features, not merely harder instances in a task-difficulty sense. We verify this by testing whether sequence length explains cluster membership for Llama3.2-1B on MathInstruct.

The less- and well-represented clusters have similar mean lengths (194 vs. 210 tokens), suggesting length is not the primary driver. We further partition examples into length bins of width 64 and compute the per-bin loss delta (upsampled minus original) separately for each cluster. Loss on less-represented examples decreases with upsampling across every length bin; for well-represented examples, upsampling *increases* loss in shorter bins ($\leq$256 tokens) and yields negligible change in longer bins. This is precisely what we would expect if the model trained on upsampled data is relying less on simple features and more on complex ones, a direct signature of reduced SB. To quantify the relative contribution of each factor, we regress per-example loss reduction on cluster membership and normalized sequence length: cluster membership explains more than twice the variance of sequence length, confirming that the clustering signal is not reducible to length.

F Detailed Ablation Results

Table 8: **Upsampling accuracy across optimizers and proxies.** Accuracy of Phi2-2.7B trained using AdamW and Muon with upsampling variants. Where compute permitted, we ran two seeds and report mean ± std (entries without ± are single runs); other settings are identical to those in Table 4.

		AdamW			Muon	
Dataset	PT	FT	Upspl. (Proxy: Phi)	Upspl. (Proxy: Pythia)	FT	Upspl.
GSM8K	56.2	72.3 ±0	72.6 ±1.6	72.7 ±1.1	72.6	**73.4 ±1.1**
MATH	16.8	33.4	**35.1 ±0.4**	34.9 ±0.1	34.3	**35.1 ±0.6**
NumGLUE	34.7	60.7	65.7 ±1.3	64.7 ±0.2	64.9	**66.0 ±0.4**
SVAMP	67.8	77.7	**78.6 ±0.5**	77.3 ±0.9	77.9	77.7 ±0.3
DeepMind	33.9	**60.2**	58.1 ±0.1	57.9 ±0.1	57.5	59.9 ±0.1
MMLU-Math	12.4	35.0	**38.3 ±0.4**	36.6 ±0.4	36.1	35.8 ±0.4
Avg.	37.0	56.5	**58.0**	57.3	57.2	**58.0**

Table 9: **Loss clustering ablations for Qwen3-0.6B-Base, Llama3.2-1B, and Phi2-2.7B across math benchmarks.** Each model was finetuned under two settings: upsampling using loss trajectory across multiple proxy model checkpoints (as described in the paper), and upsampling using loss *at* the final proxy model checkpoint. (For comparison, we also include the performance of the finetuned model (FT) without upsampling.) The highest performance *among the ablations* (within a tolerance of 1.0%) is indicated in **bold**; accuracies strictly exceeding the FT baseline are additionally underlined. All other settings (e.g., clustering model, evaluation setup) are identical to those in Table 4.

	Qwen3-0.6B-Base			Llama3.2-1B			Phi2-2.7B		
Dataset	FT	Traj.	Ckpt.	FT	Traj.	Ckpt.	FT	Traj.	Ckpt.
GSM8K	59.0	**59.6**	**59.4**	22.7	**25.6**	**25.6**	72.3	**72.5**	**72.6**
MATH	36.1	**37.9**	**37.1**	14.6	**17.7**	**17.4**	33.4	**35.0**	**35.0**
NumGLUE	62.5	**66.3**	**65.4**	31.5	**36.1**	33.5	60.7	**65.7**	**65.5**
SVAMP	75.6	**75.6**	**76.6**	41.7	40.2	**41.3**	77.7	**78.5**	75.8
DeepMind	65.4	**64.7**	**65.5**	31.4	**34.5**	**34.9**	60.2	**58.0**	56.2
MMLU-Math	28.2	30.2	**31.9**	27.2	25.8	**27.8**	35.0	**38.2**	36.1
Avg.	54.5	**55.7**	**56.0**	28.2	**30.0**	**30.1**	56.5	**58.0**	56.9

Table 10: **Upsampling ablations for Qwen3-0.6B-Base and Llama3.2-1B across math benchmarks.** Each model was finetuned under three settings: random upsampling, upsampling *well-represented* examples, and upsampling less-represented examples. We ensure that the total number of upsampled examples is identical across ablations; where needed, a random subset of examples was selected. (For comparison, we also show the finetuned model performance (FT).) The highest performance *among the ablations* (within 1.0%) is indicated in **bold**; accuracies strictly exceeding FT are additionally underlined. All other settings are identical to those in Table 4.

	Qwen3-0.6B-Base				Llama3.2-1B			
Dataset	FT	Rand.	Well	Less	FT	Rand.	Well	Less
GSM8K	59.0	56.2	55.0	**59.6**	22.7	22.4	20.4	**25.6**
MATH	36.1	36.6	36.5	**37.9**	14.6	16.1	14.9	**17.7**
NumGLUE	62.5	64.0	61.4	**66.3**	31.5	33.5	29.5	**36.1**
SVAMP	75.6	**74.7**	**75.3**	**75.6**	41.7	**40.7**	**40.2**	**40.2**
DeepMind	65.4	**65.2**	62.4	**64.7**	31.4	31.2	30.4	**34.5**
MMLU-Math	28.2	28.6	27.1	**30.2**	27.2	**25.7**	24.2	**25.8**
Avg.	54.5	54.2	53.0	**55.7**	28.2	28.3	26.6	**30.0**

Table 11: **MathInstruct benchmarking results for Phi2-2.7B under different upsampling strategies.** We compare finetuning (FT), duplication upsampling (UP), autoregressive synthesis (LLMSYN), and LLaDA synthesis. Scores within 1.0 percentage point of the best augmentation method are in **bold**; scores strictly exceeding FT are underlined.

	Phi2-2.7B			
Dataset	FT	UP	LLMSYN	LLaDA
GSM8K	72.0	**73.7**	72.1	72.3
MATH	34.3	**35.3**	33.8	**35.1**
NumGLUE	62.2	**63.4**	59.4	**62.6**
SVAMP	79.7	**79.6**	**80.2**	**80.4**
DeepMind	56.8	57.4	55.9	**60.5**
MMLU-Math	35.8	**36.3**	32.9	**35.9**
Avg.	56.8	**57.6**	55.7	**57.8**

Table 12: **Phi2-2.7B MathInstruct benchmarking at different masking ratios.** We compare finetuning (FT) with the current LLaDA results at masking ratio 0.30 and the historical results at masking ratio 0.45. Scores within 1.0 percentage point of the best augmentation method are in **bold**; scores strictly exceeding FT are underlined.

	Phi2-2.7B		
Dataset	FT	LLaDA (0.30)	LLaDA (0.45)
GSM8K	72.0	**72.3**	**71.4**
MATH	34.3	**35.1**	**34.9**
NumGLUE	62.2	62.6	**65.1**
SVAMP	79.7	**80.4**	77.9
DeepMind	56.8	60.5	**61.6**
MMLU-Math	35.8	**35.9**	32.5
Avg.	56.8	**57.8**	**57.2**

G Synthetic Augmentation

G.1 Source-Conditioned Generation with LLaDA

We augment the less-represented examples selected by the loss-trajectory procedure in Section 5. Each selected source retains its original training example and contributes one additional representative, which is either a generated problem with newly constructed supervision or, after unsuccessful generation, a duplicate of the original example. The following procedure concerns mathematical augmentation.

Problem Generation. We use frozen LLaDA-8B-Base to reconstruct partially masked problem statements. The original answer and rationale are removed before generation. For newly generated mathematical examples, we identify digit-containing tokens and form masking neighborhoods extending up to two tokens on either side. We protect the first token and tokens containing LaTeX structural characters, randomly order the candidate neighborhoods, and accumulate their union toward a target mask fraction of 0.30, stopping earlier if eligible neighborhoods are exhausted. The realized mask fraction may therefore differ from the target. Infilling uses Gumbel-Max sampling with temperature 0.7 and at most 64 denoising iterations, progressively committing the highest-confidence sampled tokens while retaining the unmasked context and previously filled positions. The model uses bfloat16 precision without classifier-free guidance.

Supervision and Acceptance. The generated problem does not inherit its source answer. GPT-OSS-20B produces three sampled solutions per candidate, using temperature 0.6 and top-p 0.95 for newly generated mathematical examples. A candidate is accepted when at least two of the three extracted final answers agree. For multiple-choice examples, GPT-OSS-20B also constructs distractor options. This agreement check filters inconsistent candidates but does not guarantee correctness.

Retry and Fallback. Each source receives at most three generation attempts in total. An unsuccessful attempt is discarded, and the next attempt starts from the original question with a fresh mask. Only sources without an accepted candidate are retried. If all attempts fail, the additional representative is the complete original problem–supervision pair. Thus generation success changes the content of the added representative while preserving the amplification weight of every selected source.

G.2 Gradient-Space Analysis

G.2.1 Setup and Conditional Decomposition

Fix model parameters θ and a unit task direction u determined independently of the augmentation draws. For a complete training example $s = (x, y)$, let $g_\theta(s) = \nabla_\theta \ell(\theta; x, y)$, and define $\Pi = I - uu^\top$. All expectations below condition on the fixed model, reference direction, and task structure. We first consider a homogeneous pool of m source examples, each contributing k representatives from the same conditional distribution. The training pool that retains an original example alongside generated representatives is treated in Appendix G.2.3.

Sampling Assumptions. The sources $s_1, \ldots, s_m$ are independent and identically distributed. Conditional on s_j, the k representatives are independent and identically distributed under $G_{\mathrm{train}}(\cdot \mid s_j)$, with independent generation randomness across sources. Both original and representative gradients have finite second moments. Here $m, k \geq 1$ are integers. The distribution G_{train} is induced by the complete augmentation procedure, including problem generation, supervision construction, validation, retries, and fallback. It can therefore place positive probability on the original example. Independent draws refer to independent runs of this complete procedure.

Hierarchical Decomposition. For representative gradient $g_{j,r}$, define

$$\begin{aligned} a_{j,r} &= \langle g_{j,r}, u \rangle, & b_G &= \mathbb{E}[\Pi g_{j,r}], \\ \beta_j &= \mathbb{E}[\Pi g_{j,r} \mid s_j] - b_G, & \eta_{j,r} &= \Pi g_{j,r} - b_G - \beta_j. \end{aligned} \tag{22}$$

Then

$$g_{j,r} = a_{j,r} u + b_G + \beta_j + \eta_{j,r}. \tag{23}$$

The residual terms are orthogonal to u, with $\mathbb{E}[\beta_j] = 0$ and $\mathbb{E}[\eta_{j,r} \mid s_j] = 0$. Thus b_G is the global mean residual, β_j is shared by representatives of the same source, and $\eta_{j,r}$ is the variant-specific fluctuation. Write

$$\mu_G = \mathbb{E}[a_{j,r}], \qquad B_G = \|b_G\|^2, \qquad \tau_G^2 = \mathbb{E}\|\beta_j\|^2, \qquad \sigma_G^2 = \mathbb{E}\|\eta_{j,r}\|^2. \tag{24}$$

Orthogonality to the chosen reference direction does not imply irrelevance to other tasks.

G.2.2 Residual Aggregation and Comparison with Duplication

Theorem 15 (Coherence hierarchy)**.** *Let $\Gamma_G^{(m,k)} = \sum_{j=1}^{m} \sum_{r=1}^{k} g_{j,r}$. Under the sampling assumptions above,*

$$\begin{aligned} \mathbb{E}\langle \Gamma_G^{(m,k)}, u \rangle &= mk\mu_G, \\ \mathbb{E}\|\Pi\Gamma_G^{(m,k)}\|^2 &= m^2k^2B_G + mk^2\tau_G^2 + mk\sigma_G^2. \end{aligned} \tag{25}$$

Proof. The first identity follows by linearity. For the second, write $\Pi\Gamma_G^{(m,k)} = mkb_G + k\sum_j \beta_j + \sum_{j,r} \eta_{j,r}$. Centering removes the cross terms with b_G. Conditional centering gives $\mathbb{E}\langle \beta_j, \eta_{j,r} \rangle = 0$, and conditional independence gives $\mathbb{E}\langle \eta_{j,r}, \eta_{j,t} \rangle = 0$ for $r \neq t$. Independence across sources removes the remaining cross-source terms. The three surviving contributions are $m^2k^2B_G$, $mk^2\tau_G^2$, and $mk\sigma_G^2$. □

Signal-to-Residual Ratio. For positive mean task alignment and positive residual energy, define

$$Q_G(m,k) = \frac{\left(\mathbb{E}\langle \Gamma_G^{(m,k)}, u \rangle\right)^2}{\mathbb{E}\|\Pi\Gamma_G^{(m,k)}\|^2} = \frac{mk\mu_G^2}{mkB_G + k\tau_G^2 + \sigma_G^2}. \tag{26}$$

Exact duplication, denoted by U, is deterministic conditional on the source, so $\sigma_U^2 = 0$ and $Q_U(m,k) = m\mu_U^2/(mB_U + \tau_U^2)$. In particular, duplication repeats the task signal and source-specific residual together, leaving this ratio unchanged as k increases.

Theorem 16 (Conditions under which generation improves the signal-to-residual ratio)**.** *Suppose $\mu_G = \mu_U > 0$, $B_G = B_U$, and both residual denominators are positive. Then*

$$Q_G(m,k) > Q_U(m,k) \quad \Longleftrightarrow \quad \sigma_G^2 < k\big(\tau_U^2 - \tau_G^2\big). \tag{27}$$

Proof. Applying Equation 26 to G and U gives ratios with the same positive numerator $mk\mu_G^2$. Using $\sigma_U^2 = 0$, comparison of their denominators gives $mkB_G + k\tau_G^2 + \sigma_G^2 < mkB_U + k\tau_U^2$. Canceling the matched drift terms yields Equation 27. □

The condition isolates the trade-off between reducing source-specific coherence and introducing variant-level variation. Matching alignment and drift is an analytical assumption, not a guaranteed property of LLaDA. Without matching, the full ratios in Equation 26 must be compared. These quantities are local gradient diagnostics; they neither equal the feature-learning-time entropy used in the SAM analysis nor establish a downstream generalization bound.

G.2.3 Retaining the Original Example

The implemented pool retains each original example and adds $n = k - 1$ outputs of the augmentation procedure, with $n = 1$ by default. The original gradient $g_j^0 = g_\theta(s_j)$ is deterministic conditional on its source and need not follow the generated distribution. Define

$$\begin{aligned} g_j^0 &= a_j^0 u + b_0 + \beta_j^0, & b_0 &= \mathbb{E}[\Pi g_j^0], \\ \mu_0 &= \mathbb{E}[a_j^0], & \tau_0^2 &= \mathbb{E}\|\beta_j^0\|^2, \\ \beta_j^0 &= \Pi g_j^0 - b_0, & C_{0G} &= \mathbb{E}\langle \beta_j^0, \beta_j \rangle. \end{aligned} \tag{28}$$

Here $a_j^0 = \langle g_j^0, u \rangle$, and C_{0G} is a signed covariance that need not vanish. Exact duplication has $\mu_U = \mu_0$, $B_U = \|b_0\|^2$, and $\tau_U^2 = \tau_0^2$.

For $\Gamma_{\text{mix}}^{(m,k)} = \sum_{j=1}^{m}(g_j^0 + \sum_{r=1}^{n} g_{j,r})$, the same independent-source assumptions give

$$\begin{aligned}\mathbb{E}\langle \Gamma_{\text{mix}}^{(m,k)}, u\rangle &= m(\mu_0 + n\mu_G),\\ \mathbb{E}\|\Pi\Gamma_{\text{mix}}^{(m,k)}\|^2 &= m^2\|b_0 + nb_G\|^2 \\ &\quad + m\big(\tau_0^2 + n^2\tau_G^2 + 2nC_{0G} + n\sigma_G^2\big).\end{aligned} \tag{29}$$

Proof. Expand the projected gradient as $m(b_0 + nb_G) + \sum_j(\beta_j^0 + n\beta_j + \sum_r \eta_{j,r})$. Conditional centering removes cross terms involving $\eta_{j,r}$, and independence removes cross-source terms. Within a source, the original and generated source effects contribute $\mathbb{E}\|\beta_j^0 + n\beta_j\|^2 = \tau_0^2 + n^2\tau_G^2 + 2nC_{0G}$. Adding the n variant fluctuations proves Equation 29. □

For $\mu_0 + n\mu_G > 0$ and positive residual energy, the corresponding ratio is

$$Q_{\text{mix}}(m,k) = \frac{m(\mu_0 + n\mu_G)^2}{m\|b_0 + nb_G\|^2 + \tau_0^2 + n^2\tau_G^2 + 2nC_{0G} + n\sigma_G^2}. \tag{30}$$

Under $\mu_G = \mu_0 > 0$, $b_G = b_0$, positive comparison denominators, and $n \geq 1$, comparison with $Q_U = m\mu_0^2/(m\|b_0\|^2 + \tau_0^2)$ gives

$$Q_{\text{mix}}(m,k) > Q_U(m,k) \quad \Longleftrightarrow \quad \sigma_G^2 < n(\tau_0^2 - \tau_G^2) + 2(\tau_0^2 - C_{0G}). \tag{31}$$

With matched alignment and drift, the comparison is equivalent to $\tau_0^2 + n^2\tau_G^2 + 2nC_{0G} + n\sigma_G^2 < (n+1)^2\tau_0^2$. Rearranging and dividing by $n > 0$ proves Equation 31. For the default $n = 1$, the condition becomes

$$\tau_G^2 + \sigma_G^2 + 2C_{0G} < 3\tau_0^2. \tag{32}$$

This extension accounts explicitly for dependence on the retained original. Unlike the homogeneous comparison, its matched-drift simplification requires $b_G = b_0$, not merely equality of their norms. The homogeneous condition in Equation 27 should therefore not be substituted directly for the condition on the implemented pool.

G.3 Details of generation

The two synthesis procedures use different conditioning mechanisms. LLMSYN receives the original question and its reference solution and is instructed to generate a different question compatible with that solution. LLaDA instead infills a partially masked original question; the original solution is not provided to the infilling model. A separate autoregressive solver generates a solution to the resulting question.

LLMSYN: Solution-Conditioned Question Synthesis. Our LLMSYN implementation uses GPT-OSS-20B (Agarwal et al., 2025) with the following system message and user template:

Prompt for solution-conditioned question synthesis

```
Your task is to create a NEW math problem that is:

  • DIFFERENT from the reference problem (different scenario,
    wording, and approach)
  • At the SAME difficulty level as the reference problem
  • Solvable with the EXACT same solution shown below:
    <reference_problem>
    {orig_question}
    </reference_problem>

    <target_solution>
    {original_solution}
    </target_solution>

STEP 1 -- Write a new problem:
Design a problem that naturally leads to the target solution.
Match the type:
```

```
  • Numbers/values → "Calculate", "Find", "Determine"
  • Equations → "Simplify", "Solve for", "Express"
  • Code → A realistic scenario requiring a program (do NOT say
    "write code for this solution")
  • Proofs → "Prove that", "Show that"

Requirements:

  • Include all information needed to solve it
  • Difficulty must match the reference (same depth of reasoning,
    similar number of steps)
  • Do NOT copy the reference problem's scenario or phrasing

STEP 2 -- Verify:
Solve your new problem step-by-step and confirm the answer matches
the target solution exactly.

STEP 3 -- Check:

  • Same final answer?
  • Same difficulty level?
  • Genuinely different from the reference?

If any check fails → revise and repeat from Step 1.

FINAL OUTPUT (only after all checks pass):
<question>
[Your new problem -- do NOT include the solution]
</question>

<verification>
[One sentence:  "Verified -- solving gives {original_solution}"]
</verification>
```

LLaDA: Masked-Question Infilling. LLaDA-8B-Base does not receive a natural-language rewriting instruction or a chat wrapper. Its input is the tokenized original question with selected token positions replaced by the model's mask token. The following box is a schematic representation of this input, not a literal textual prompt:

LLaDA input (schematic)

```
{masked_original_question}
```

$$\widetilde{q} = \mathrm{Mask}(q, M), \qquad q' = \mathrm{Infill}_{\mathrm{LLaDA}}(\widetilde{q}).$$

The retained implementation selects spans around tokens containing digits, with a context radius of two tokens and a target mask ratio of 0.30. Tokens containing LaTeX structural characters are protected. The realized mask ratio can differ from the target because masking is span-based. Infilling uses 64 denoising steps and temperature 0.7; unmasked tokens remain fixed. The original solution is not part of this input.

Solution Generation Following LLaDA. The retained solver implementation uses GPT-OSS-20B. It supplies a single user message, with no explicitly supplied system message, and chooses the suffix according to the source's CoT/PoT annotation. Here, `{question}` denotes the generated question; for multiple-choice questions, the existing choices are removed before solving the question stem.

GPT solver prompt after LLaDA: CoT

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

GPT solver prompt after LLaDA: PoT

```
{question}
Solve this by writing a short Python program (in a ```python code
block) that computes and prints the answer, then state the final
numeric answer in \boxed{}.
```

The retained gate samples three solutions with temperature 0.6, top-$p = 0.95$, a maximum of 1,600 output tokens, and low reasoning effort. A candidate is retained when at least two samples agree on the parsed and normalized final answer. PoT solutions must also contain a code block. This is a self-consistency filter, not an independent proof of correctness.

Multiple-Choice Distractors in the Reused v3 Branch. The Phi-2 LLaDA training data combine reused v3 examples with newly generated examples. The following additional user template belongs to the reused v3 multiple-choice branch; it is not applied uniformly to all examples. GPT-OSS-20B receives the original multiple-choice question (truncated to 1,500 characters) as `{orig}` and the solver-derived answer as `{ans}`.

GPT distractor prompt: reused v3 MC branch

```
Original multiple-choice math problem (style reference):
{orig}
A new, similar problem has this correct answer: {ans}
Write exactly 4 plausible but INCORRECT answer options for the new
problem. Match the style, number format, units, precision and
spread of the ORIGINAL problem's answer choices above. None of
them may equal the correct answer. Output ONLY the 4 options, one
per line, no letters, no explanations.
```

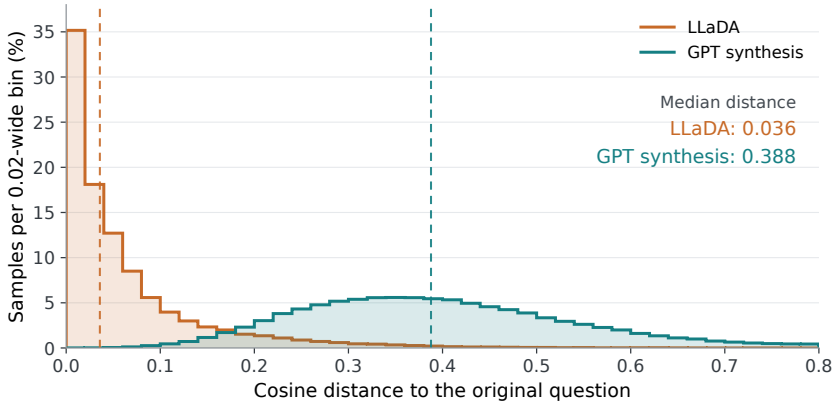

Figure 9: **Locality of synthetic math questions.** Distribution of cosine distances between generated questions and their corresponding originals in the historical generation pools. LLaDA (orange) produces questions closer to their sources in embedding space than GPT-based synthesis (teal). Dashed lines indicate the median distances: 0.036 for LLaDA and 0.388 for GPT-based synthesis. Each bin spans a distance of 0.02.

Locality of synthetic questions. Figure 9 compares the embedding distance between generated questions and their corresponding originals. LLaDA produces substantially more local variations,

with a median cosine distance of 0.036, compared with 0.388 for GPT-based synthesis. This suggests that diffusion-based infilling introduces changes while remaining closer to the source question in embedding space. A plausible hypothesis is that such local variations preserve useful properties of the selected training examples while introducing diversity, which may help explain the favorable LLaDA results in Table 11.

Table 13 gives an example of different synthetic methods.

Table 13: **An example of LLMSYN and LLADA synthesis.** Both synthetic questions originate from the same MathInstruct training example. LLMSYN changes the scenario while retaining the original solution, whereas LLaDA changes the numerical quantities. The solution for the LLaDA variant is generated and filtered by GPT-OSS-20B. Solution wording is retained, with mathematical formatting normalized.

Method	Question	Corresponding solution
Original	John buys 2 dozen cookies. He eats 3. How many cookies does he have left?	He got $2 \times 12 = 24$ cookies. So he has $24 - 3 = 21$ cookies left. The answer is 21.
LLMSYN	Samantha orders 2 dozen cookies for a party. During the preparation she eats 3 of them. How many cookies are left to serve?	He got $2 \times 12 = 24$ cookies. So he has $24 - 3 = 21$ cookies left. The answer is 21.
LLaDA	John buys **4** dozen cookies. He eats **6**. How many cookies does he have left?	John buys 4 dozen cookies. 4 dozen $= 4 \times 12 = 48$ cookies. He eats 6 cookies, so the number remaining is 48 - 6 = 42. \boxed{42}

H Qualitative Characterization of Trajectory Clusters

To complement our theoretical and quantitative analyses, we manually inspect training instances assigned to the well-represented (low early-loss trajectory) and less-represented (high early-loss trajectory) clusters on the MathInstruct dataset. Representative examples and structural annotations are presented in Table 14.

Table 14: **Qualitative comparison of clusters identified on MathInstruct.** The well-represented cluster is dominated by repetitive, templated multiple-choice arithmetic, whereas the less-represented cluster captures under-represented reasoning modalities (e.g., Python code rationales) and unconstrained multi-step derivations.

Cluster (pattern)	Representative example	Annotation
Well-represented (templated multiple-choice arithmetic)	**Q:** "Sides of a rectangular park are in the ratio $3:2$ and its area is 3750 sq m, the cost of fencing it at 50 ps per meter is? Answer Choices: (A) Rs. 122 (B) Rs. 129 (C) Rs. 125…" **A:** "Let's solve the multi-choice question step by step. $3x \cdot 2x = 3750 \implies x = 25$. $2(75 + 50) = 250$m. $250 \cdot 1/2 =$ Rs. 125. The answer is C."	High-frequency multiple-choice scaffold with rigid opening/closing boilerplate and compressed single-line equations. Rapidly fit by early gradient steps.
Less-represented (free-form derivation)	**Q:** "Let $F_1 = (0, 1)$ and $F_2 = (4, 1)$. Then the set of points P such that $PF_1 + PF_2 = 6$ form an ellipse. The equation of this ellipse can be written as $\frac{(x-h)^2}{a^2} + \ldots$" **A:** "We have that $2a = 6$, so $a = 3$. The distance between the foci is $2c = 4$, so $c = 2$. Hence $b = \sqrt{a^2 - c^2} = \sqrt{5}\ldots$"	Unconstrained multi-step analytical derivation without multiple-choice options or templated framing. Requires multi-hop symbolic chaining.
Less-represented (programmatic rationale)	**Q:** "At a pool party, there are 4 pizzas cut into 12 slices each. If guests eat 39 slices, how many slices are left? Let's write a Python program." **A:** `total_slices = 4 * 12; eaten_slices = 39; left_slices = total_slices - eaten_slices`	Elementary arithmetic difficulty, but expressed as a Python program (minority rationale format) rather than dominant natural-language CoT prose.

Our inspection reveals that early-training loss trajectories partition the dataset by structural and reasoning-modality representation rather than raw mathematical difficulty alone. Specifically, the well-represented cluster contains examples whose loss drops rapidly during the first $\frac{1}{3}$ of an epoch, predominantly consisting of templated, multiple-choice arithmetic word problems. These instances share a highly repetitive prompt scaffold ("Answer Choices: (A)…(B)…") and a standardized opening or closing rationale boilerplate ("Let's solve the multi-choice question step by step…The answer is [X]"). Because gradient descent quickly fits these high-frequency surface tokens, the model achieves low loss early in training without necessarily mastering complex reasoning.

Conversely, the less-represented cluster contains examples that maintain a higher loss across early checkpoints, falling into two distinct categories of under-representation. The first comprises free-form multi-step derivations, such as analytical geometry or algebraic proofs, which lack multiple-choice options and require constructing an unconstrained symbolic derivation path. The second consists of programmatic rationales, where the problem itself may be elementary arithmetic (e.g., a GSM-style word problem), but the target supervision is expressed as executable Python code rather than natural-language prose. Because code-based rationales represent a smaller fraction of the overall corpus relative to natural-language chain-of-thought (CoT), their governing features are learned much later in training.

I Broader Impacts and Limitations

Broader Impacts. Our work advances the theoretical understanding of simplicity bias in Transformer-based LLMs, and uses these insights to propose an optimizer-agnostic data-centric approach for improving LLM generalization. The data-selection step requires per-example loss tracking via a small proxy model, and the resulting augmented dataset can be integrated into existing LLM finetuning workflows without architectural changes or modified loss functions. Additionally, since our approach complements existing data-centric techniques such as curriculum learning and OHEM, it can be readily stacked with them for further gains. More broadly, our work links optimization dynamics to feature learning behavior, and therefore provides a foundation for developing methods that systematically control inductive biases in LLMs, which we hope will inspire further investigation into the role of data distribution in shaping generalization.

Improving LLM generalization can benefit downstream applications. However, since our method amplifies less-represented examples based on loss dynamics rather than semantic content, it may inadvertently reinforce latent biases present in the training data, for instance, if certain demographic groups are systematically less represented in the feature space of the proxy model. We encourage practitioners to audit the clusters for such disparities before deployment in sensitive applications.

Limitations. Our theoretical analysis uses a simplified multi-head linear self-attention model with Gaussian inputs and vanishing initialization, abstracting away the nonlinear and multilayer structure of modern Transformers. These assumptions are consistent with prior theoretical analyses of Transformer learning dynamics (Zhang et al., 2025) and enable a tractable analysis of stage-wise feature learning, while our practical data-centric method does not rely on these theoretical idealizations and is evaluated directly on pretrained nonlinear LLMs.

Our empirical pipeline requires a lightweight proxy model to profile early loss trajectories, introducing a one-time preprocessing step with a small compute overhead ($\approx 0.3\%$ of standard fine-tuning). Ablations show that the identified examples and resulting gains are stable across proxy capacities and trajectory lengths (Figure 7). Finally, our experiments focus on SFT and do not analyze how upsampling or generation affects other post-training stages such as RL, which we leave as future work.