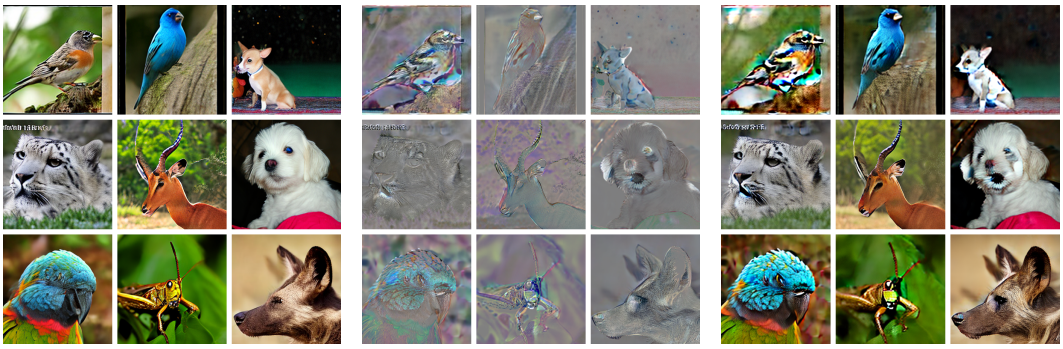
IMPROVING GENERATIVE MODEL SELF-TRAINING WITH GEOMETRICALLY MODIFIED OUTPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-training generative models – the continued improvement of a model using its own outputs – is becoming increasingly important as high-quality training data becomes scarce. However, naïvely finetuning on model-generated samples leads to degradation through model collapse and the model autophagy disorder. Negative-guidance self-training methods turn this degradation into a useful signal, using a model finetuned on its own outputs to guide the original model toward improved generation. Existing methods, however, take the negative signal in standard model outputs as given. We instead ask whether this signal can be explicitly strengthened. We introduce Geometrically Modified Outputs (GMOs), which reweight the singular values of the generators input-output Jacobian to increase the influence of its leading singular directions. This geometric modification amplifies the mode-seeking behavior and distortions of standard outputs, providing a stronger and more targeted negative signal for self-training. Across a range of one-step generative models, GMOs consistently improve the performance of negative-guidance methods, including Neon and SIMS, compared with using standard model outputs.



Standard Outputs | Perturbation | Geometrically Modified Outputs

Figure 1: **Geometrically Modified Outputs (GMOs) improve the performance of one-step generative models through self-training.** Here we present examples of GMOs for the IMM (Zhou et al., 2025) one-step generative model trained on Imagenet256 (Krizhevsky et al., 2012). In the first panel, we show the model's standard outputs; in the center panel, we show the perturbations of these outputs that yield the corresponding GMOs in the right panel (see Figure 11 for more examples). Using GMOs in self-training algorithms more effectively improves generative models.

1 INTRODUCTION

The performance of generative models for image generation has benefited greatly from increasing amounts of compute and high-quality training data (Kaplan et al., 2020; Henighan et al., 2020). Although the path for increasing compute is relatively clear (AI, 2026), significantly expanding the quantity of high-quality training data represents a fundamental challenge (Muennighoff et al., 2025; Villalobos et al., 2024). Consequently, the field of self-training–the continued development of a model using only its own outputs–is gaining significant interest (Kim et al., 2023; Alemohammad

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.

et al., 2024b; Yuan et al., 2024; Alemohammad et al., 2026; Zheng et al., 2025; Feng et al., 2025; Karras et al., 2024).

Although the utilization of synthetic data has been established in adjacent fields, such as for data augmentation when training classifiers (Wang et al., 2023; He et al., 2023) and for bootstrapping reinforcement learning models through self-play (Silver et al., 2018), the situation is more nuanced with generative models. Naïve self-training–finetuning a generative model on its own outputs–is detrimental as it leads to the "model autophagy disorder" (MAD) (Alemohammad et al., 2024a) and "model collapse" (Shumailov et al., 2024). These phenomena describe the degradation of model quality due to the appearance of unwanted artifacts in its outputs and a reduction in the diversity of those outputs. Common strategies implemented by self-training algorithms include, self-play (Yuan et al., 2024), direct discriminative optimization guidance (Zheng et al., 2025), and output verification (Feng et al., 2025), with the strongest results observed using negative guidance (Alemohammad et al., 2024b; 2026; Karras et al., 2024).

In this paper, we ask: *Can the effectiveness of self-training algorithms using negative guidance be improved by augmenting model outputs to amplify the negative signal?*

We answer this question affirmatively with Geometrically Modified Outputs (GMOs), by demonstrating that they can be used to improve the effectiveness of the SIMS (Alemohammad et al., 2024b) and Neon self-training algorithm (Alemohammad et al., 2026).

GMOs leverage the observation of Batsell et al. (2026), and verified in Figure 2, that the effective ranks of the input-output Jacobians of generative models collapse during naïve self-training. Examples of how GMOs compare with standard model outputs are shown in Figures 1 and 11. GMOs represent a geometric perspective on the self-training problem, which has not been explicitly leveraged in prior works (Kim et al., 2023; Alemohammad et al., 2024b; Yuan et al., 2024; Alemohammad et al., 2026; Zheng et al., 2025; Feng et al., 2025; Karras et al., 2024).

2 The Geometry and Self-training of Generative Models

In this section, we introduce relevant notation for the generative models we consider (Section 2.1), we describe what we mean by the *geometry* of generative models (Section 2.2), and we introduce the *self-training* problem (Section 2.3).

2.1 Generative Models

We consider generative models G_θ with parameters θ that are resultant of a training algorithm $\mathcal{A}$ being applied to a set of training data $\mathcal{D}$ drawn from a distribution p_{data}. In this context, $\mathcal{D}$ represents high-quality real data. Generative model architecture can differ in the inference routines they use for generating outputs. Generally, we let $\mathcal{I}$ represent this inference routine and let $q_{\theta,\kappa}$ denote the induced sampling distribution when $\mathcal{I}$ is implemented with hyperparameter κ.

In this paper, we are concerned with one-step inference routines (Song et al., 2023; Frans et al., 2025; Geng et al., 2025; Zhou et al., 2025; Deng et al., 2026; Yue et al., 2026; Yang et al., 2026). More specifically, $\mathcal{I}$ generates an output $\boldsymbol{s} \in \mathbb{R}^d$ based on a latent vector $\boldsymbol{z} \in \mathbb{R}^h$, which we summarize as a map $g_\theta : \mathbb{R}^h \to \mathbb{R}^d$.

2.2 The Geometry of Generative Models

A standard output of a one-step generative model is generated by sampling a latent vector $\boldsymbol{z} \in \mathbb{R}^h$ and applying the map $g_\theta(\boldsymbol{z}) = \boldsymbol{s} \in \mathbb{R}^d$. We can decompose this output as $\boldsymbol{s} = \boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z} + \boldsymbol{b}_{\boldsymbol{z}}$ where $\boldsymbol{J}_{\boldsymbol{z}} \in \mathbb{R}^{d\times h}$ is the input-output Jacobian of g_θ at $\boldsymbol{z}$, and $\boldsymbol{b}_{\boldsymbol{z}} := \boldsymbol{s} - \boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z}$ is the offset of g_θ at $\boldsymbol{z}$. We take the *local geometry* of g_θ at $\boldsymbol{z}$ to be the spectral properties of $\boldsymbol{J}_{\boldsymbol{z}}$. Let $\boldsymbol{u}_{\boldsymbol{z}}^{(k)} \in \mathbb{R}^d$ and $\boldsymbol{v}_{\boldsymbol{z}}^{(k)} \in \mathbb{R}^h$ denote the kth left and right singular vectors of $\boldsymbol{J}_{\boldsymbol{z}}$, respectively, with $\sigma_{\boldsymbol{z}}^{(k)} \in \mathbb{R}$ being the corresponding singular value. Let $\boldsymbol{U}_{\boldsymbol{z}} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{V}_{\boldsymbol{z}} \in \mathbb{R}^{h\times r}$ be the corresponding matrices of left and right singular vectors, such that $\boldsymbol{J}_{\boldsymbol{z}} = \boldsymbol{U}_{\boldsymbol{z}}\text{diag}\left(\sigma_{\boldsymbol{z}}^{(1)}, \ldots, \sigma_{\boldsymbol{z}}^{(r)}\right)\boldsymbol{V}_{\boldsymbol{z}}^\top$.

2.3 Self-training Generative Models

Self-training is the problem of constructing a learning algorithm that uses only the model's outputs to yield a generative model $G_{\theta'}$ that outperforms G_θ. More formally, self-training involves applying a learning algorithm $\tilde{\mathcal{A}}$ on a set of data $\mathcal{S}$ sampled from $q_{\theta,\kappa}$ to identify a set of parameters θ' such that $G_{\theta'}$ performs better than G_θ. Typically, self-training algorithms are constrained by a compute budget $\mathcal{B}$, which we measure as the cumulative number of images observed when using $\tilde{\mathcal{A}}$.

Many approaches exist for tackling the self-training problem. The naïve approach is to continue the learning algorithm $\mathcal{A}$ on $\mathcal{S}$; however, several studies observe that this leads to the degradation of the generative model (Shumailov et al., 2024; Alemohammad et al., 2024a). Principled methods for overcoming this, include self-play (Yuan et al., 2024), direct discriminative optimization guidance (Zheng et al., 2025), negative guidance (Alemohammad et al., 2024b; 2026; Karras et al., 2024), and output verification (Feng et al., 2025).

In this paper, we consider whether *the effectiveness of self-training algorithms using negative guidance can be improved by augmenting model outputs to amplify the negative signal.*

In particular, we focus on the Neon self-training algorithm (Alemohammad et al., 2026), but we also consider SIMS (Alemohammad et al., 2024b). We focus on Neon as it has been demonstrated to efficiently and effectively improve the performance of state-of-the-art generative models more effectively than other self-training algorithms utilizing negative guidance (Alemohammad et al., 2026). Neon works by finetuning a base model G_θ on $\mathcal{S}$ to get $G_{\tilde{\theta}}$. It then uses the direction between θ and $\tilde{\theta}$ as a negative signal to generate parameters $\theta' = (1+w)\theta - w\tilde{\theta}$ for some $w \in \mathbb{R}^+$. This process is summarized in Algorithm 1.

Algorithm 1 Neon

Require: Base model G_θ, Inference routine $\mathcal{I}$ with hyperparameters κ, Weight-merging parameter w, Training budget $\mathcal{B}$

1: Sample $\mathcal{S}$ from $q_{\theta,\kappa}$.
2: $G_{\tilde{\theta}} \leftarrow \mathrm{FineTune}\,(G_\theta, \mathcal{S}, \mathcal{B})$
3: $\theta' \leftarrow (1+w)\theta - w\tilde{\theta}$

Neon's guarantee that this extrapolation improves the model relies on $\mathcal{S}$ being *mode-seeking*: concentrated toward the high-density regions of the model's own distribution (Alemohammad et al., 2026).

3 Geometrically Modified Outputs

We propose Geometrically Modified Outputs (GMOs) as a strategy for amplifying the mode-seeking nature of model outputs with the intention of improving negative guidance self-training algorithms. GMOs do this by amplifying the influence of the top singular vectors of the model's generator map. This is motivated by the observations of Batsell et al. (2026) which show that under naïve self-training, the input-output Jacobians of generators become increasingly collapsed. More specifically, their effective ranks[1] collapse and low-quality artifacts appear in the more prominent singular vectors. In Figure 2, we verify the collapse of the effective rank for a MeanFlow generative model (Geng et al., 2025) naïvely self-training on ImageNet256 (Krizhevsky et al., 2012). Moreover, we verify that GMOs exhibit amplified mode-seeking in Section B.

[1]The effective rank of a matrix is defined as the exponential of the entropy of its singular values.

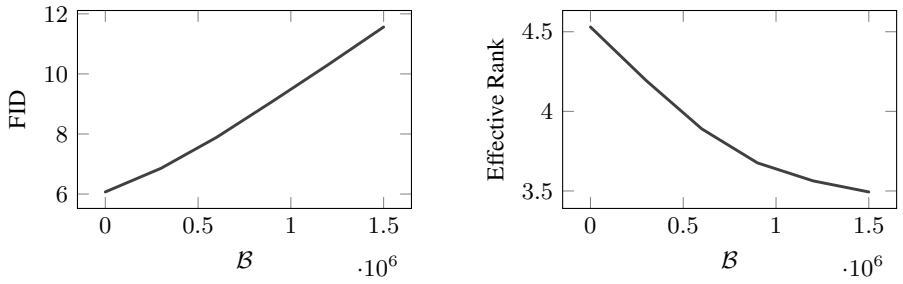


Figure 2: **The geometry of a generative model collapses under naïve self-training.** Here we finetune a MeanFlow SiT-B/2 model pre-trained on ImageNet256 for 50 epochs on $30,000$ of its own outputs. Throughout training, we monitor the FID (Heusel et al., 2017) of the model (left), and its effective rank on a collection of 16 fixed latent vectors (right).

3.1 GMO Generation

Consider a standard output $\boldsymbol{s} = \boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z} + \boldsymbol{b}_{\boldsymbol{z}} \in \mathbb{R}^d$ generated from a latent vector $\boldsymbol{z} \in \mathbb{R}^h$. The corresponding α-GMO is given by $\tilde{\boldsymbol{s}} = \tilde{\boldsymbol{J}}_{\boldsymbol{z}}\boldsymbol{z} + \boldsymbol{b}_{\boldsymbol{z}}$, where $\tilde{\boldsymbol{J}}_{\boldsymbol{z}} = \boldsymbol{U}\mathrm{diag}\left(\tilde{\sigma}_{\boldsymbol{z}}^{(1)}, \ldots, \tilde{\sigma}_{\boldsymbol{z}}^{(r)}\right)\boldsymbol{V}_{\boldsymbol{z}}^\top$ with

$$\tilde{\sigma}_{\boldsymbol{z}}^{(k)} = \begin{cases} \sqrt{(1-\alpha)\left(\sigma_{\boldsymbol{z}}^{(1)}\right)^2 + \alpha\left\|\boldsymbol{J}_{\boldsymbol{z}}\right\|_F^2} & k = 1 \\ \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(k)} & k \geq 2. \end{cases}$$

This reweighting preserves total spectral energy while transferring an α fraction of the trailing energy to the leading component (Appendix A.1).

Intuitively, the α-GMO increases the influence that the top singular vectors have on the generated output. With α equal to zero, the generated output remains unchanged, but with α equal to one, the output is entirely determined by the top singular vectors. From the specific perspective of improving Neon, GMOs can be thought of as amplifying the "mode-seeking" nature of model outputs. We explore this in Section B.

With Algorithm 2, we provide an exact implementation of α-GMOs that requires only the computation of the top spectral statistics of $\boldsymbol{J}_{\boldsymbol{z}}$.

Theorem 1. *For a generator $g : \mathbb{R}^h \to \mathbb{R}^d$, a vector $\boldsymbol{z} \in \mathbb{R}^h$ and $\alpha \in [0, 1]$, Algorithm 2 returns the α-GMO of $\boldsymbol{s} = g(\boldsymbol{z})$.*

Proof. See Section A. □

Algorithm 2 Geometrically Modifying Outputs

Require: Generator $g : \mathbb{R}^h \to \mathbb{R}^d$, $\alpha \in [0, 1]$.
1: Sample $\boldsymbol{z} \in \mathbb{R}^h$
2: $\boldsymbol{s} \leftarrow g(\boldsymbol{z})$
3: $\boldsymbol{b}_{\boldsymbol{z}} \leftarrow \boldsymbol{s} - \boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z}$
4: $(\boldsymbol{u}, \boldsymbol{v}, \sigma) \leftarrow \mathrm{TopSigVec}(g, \boldsymbol{z})$
5: $E \leftarrow \left\|\boldsymbol{J}_{\boldsymbol{z}}\right\|_F^2$
6: $\tilde{\sigma} \leftarrow \sqrt{(1-\alpha)\sigma^2 + \alpha E}$
7: $\tilde{\boldsymbol{s}} \leftarrow \sqrt{1-\alpha}\left(\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{z}}\right) + \left(\tilde{\sigma} - \sqrt{1-\alpha}\sigma\right)\left(\boldsymbol{v}^\top\boldsymbol{z}\right)\boldsymbol{u} + \boldsymbol{b}_{\boldsymbol{z}}$

3.2 Computational Complexity of Generating GMOs

Computing exact GMOs involves manipulations involving the input-output Jacobian $\boldsymbol{J}_{\boldsymbol{z}} \in \mathbb{R}^{d\times h}$. For high-dimensional generative models, where both the latent dimension h and data dimension d are large, this poses a severe computational bottleneck.

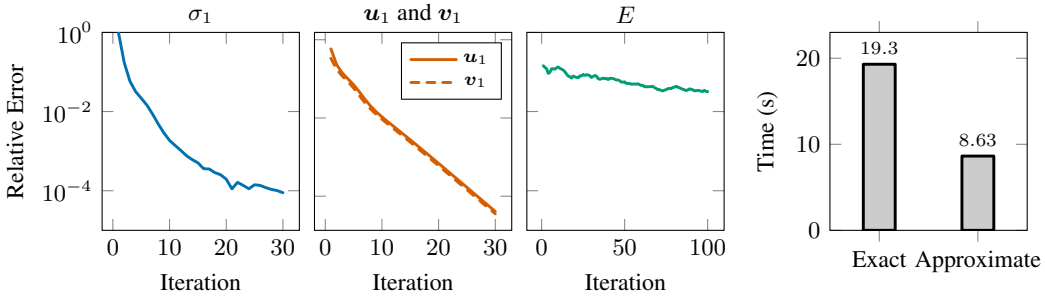


Figure 3: Here we consider computing spectral statistics from the input-output Jacobians of a MeanFlow one-step generator (Geng et al., 2025) with a UNet architecture (Ronneberger et al., 2015; Song & Ermon, 2019) trained on CIFAR10 (Krizhevsky & Hinton, 2009). Ten latent vectors are sampled, and E, σ_1, $\boldsymbol{u}_1$, and $\boldsymbol{v}_1$ are either computed exactly or using approximations. Power iteration is implemented 30 times to obtain estimates for $\hat{\sigma}_1$, $\hat{\boldsymbol{u}}_1$, and $\hat{\boldsymbol{v}}_1$. Similarly, 100 samples are used to generate the Hutchinson estimate $\hat{E}$. In the first panel, we record $|\hat{\sigma}_1 - \sigma_1|/\sigma_1$. In the second panel, we record $1 - \langle \hat{\boldsymbol{u}}_1, \boldsymbol{u}_1 \rangle / \|\hat{\boldsymbol{u}}_1\|_2 \|\boldsymbol{u}_1\|_2$, and similarly for $\boldsymbol{v}_1$. In the third panel, we record $|\hat{E} - E|/E$. In the fourth panel, we record the time taken to exactly compute the Jacobian and its singular value decomposition (Exact) and the time taken to perform 20 power iterations and 20 Hutchinson approximations (Approximate).

Here, we provide a breakdown of the computational complexity of exactly implementing Algorithm 2 and compare it with an approximate implementation using power iteration and Hutchinson's estimator. A point to note is that, this analysis is only applicable once for a given model, as given a latent vector $\boldsymbol{z}$, lines 6 and 7 of Algorithm 2 can be applied to multiple different α values to generate GMOs for different α values simultaneously.

Let $\mathcal{O}(C_f)$ denote the time complexity of a single forward pass of the generator $g(\boldsymbol{z})$.

Exact Implementation. To compute GMOs exactly, one must instantiate the full Jacobian matrix $\boldsymbol{J}_{\boldsymbol{z}}$. Using automatic differentiation, this requires either h forward-mode passes (Jacobian-Vector Products, JVPs) or d reverse-mode passes (Vector-Jacobian Products, VJPs). This scales linearly with the dimensions, resulting in a time complexity of $\mathcal{O}(\min(d, h) \cdot C_f)$. The memory complexity to store the dense matrix is $\mathcal{O}(d \cdot h)$. Once $\boldsymbol{J}_{\boldsymbol{z}}$ is materialized, computing the exact singular value decomposition to find $\boldsymbol{u}$, $\boldsymbol{v}$, and σ requires $\mathcal{O}\left(\min\left(d^2 h, dh^2\right)\right)$ operations. Computing the exact Frobenius norm requires an additional $\mathcal{O}(d \cdot h)$ operations.

In total, the exact time complexity is on the order of $\mathcal{O}\left(\min(d, h) \cdot C_f + \min\left(d^2 h, dh^2\right)\right)$, with spatial memory scaling at $\mathcal{O}(d \cdot h)$.

Approximate Implementation. To improve the tractability of computing GMOs, we can sidestep the materialization of $\boldsymbol{J}_{\boldsymbol{z}}$ entirely. Modern deep learning frameworks support fast JVPs and VJPs in roughly $\mathcal{O}(C_f)$ time, with spatial memory comparable to that of a standard forward or backward pass.

For computing approximations for $\boldsymbol{u}$, $\boldsymbol{v}$ and σ, we can use power iteration. Power iteration works by repeatedly applying the matrix $\boldsymbol{J}_{\boldsymbol{z}}^\top \boldsymbol{J}_{\boldsymbol{z}}$ to a random initialization vector. In practice, evaluating $\boldsymbol{J}_{\boldsymbol{z}}^\top (\boldsymbol{J}_{\boldsymbol{z}} \boldsymbol{w})$ amounts to one JVP followed by one VJP. For k iterations, the time complexity is $\mathcal{O}\left(k \cdot C_f\right)$.

For computing an approximate value of the squared Frobenius norm, we can use Hutchinson's estimator. The squared Frobenius norm equals the trace of the Gram matrix $\|\boldsymbol{J}_{\boldsymbol{z}}\|_F^2 = \operatorname{tr}(\boldsymbol{J}_{\boldsymbol{z}}^\top \boldsymbol{J}_{\boldsymbol{z}})$. Hutchinson's trace estimator approximates this via $\frac{1}{m} \sum_{i=1}^{m} \|\boldsymbol{J}_{\boldsymbol{z}} \boldsymbol{w}_i\|_2^2$, where $\boldsymbol{w}_i$ are standard normal vectors. Calculating this requires m independent JVPs, yielding a time complexity of $\mathcal{O}\left(m \cdot C_f\right)$.

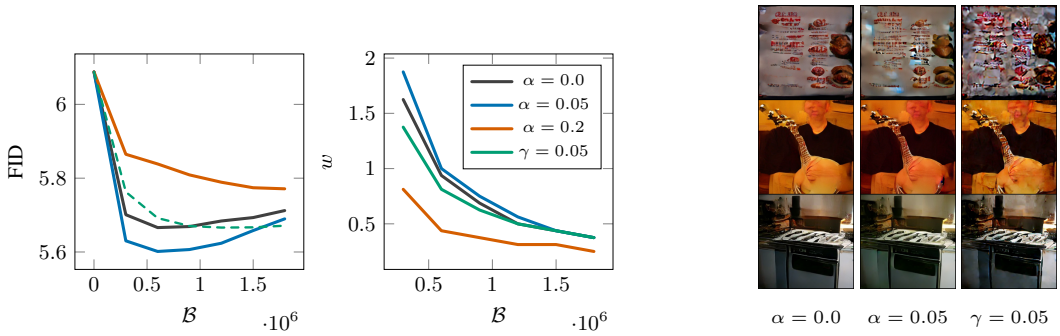


Figure 4: **GMOs provide meaningful perturbations to a model's outputs that improve the effectiveness of Neon.** Here, we consider applying the Neon self-training algorithm to the MeanFlow (Geng et al., 2025) SiT-B/2 model (Ma et al., 2024) trained on ImageNet256 (Krizhevsky et al., 2012). We generated collections of $30,000$ GMOs for alpha values 0.0 (standard outputs), 0.05, and 0.2. We then finetuned the pre-trained checkpoint with a maximum compute budget of 1.8×10^6. At regular checkpoints, we apply Neon with different weight-merging parameters w in the range $[0, 2]$. In the first and second panels, we show the minimum FID value achieved at each finetuning checkpoint with the corresponding weight-merging parameter w, respectively. In the third panel, we visually compare standard outputs to GMOs and the Gaussian baseline (see Figure 15 for more examples).

Consequently, the total time complexity for computing GMOs reduces to $\mathcal{O}\left((k+m) \cdot C_f\right)$. Similarly, the memory complexity drops to $\mathcal{O}(d+h)$, since we only have to store the vectors involved in the JVP or VJP computations.

The computational efficiency of the approximations is highlighted in the fourth panel of Figure 3, which shows that the approximate implementation of Algorithm 2 is around 2 times faster than its exact implementation. In particular, it makes the computation of GMOs tractable in settings where memory constraints would otherwise prevent it.

Importantly, with the three panels of Figure 3, we ensure that we do not lose much by utilizing this approximate implementation. The method is therefore insensitive to the power-iteration count above roughly ten iterations, and we use 20. In Figure 13, we qualitatively demonstrate that these approximations are sound by visually comparing the exact and approximate left singular vectors.

4 Experiments

In this section, we examine the characteristics of GMOs in comparison to standard outputs and how Neon with GMOs differs from Neon using standard outputs.

4.1 GMOs Outperform Standard Outputs and Gaussian Perturbed Outputs

Here, we empirically verify that applying the Neon self-training to GMOs is more effective than applying it to standard outputs. In particular, we isolate this benefit to the geometric structure of GMOs rather than to the mere effect of injecting noise. We do so by applying Neon to standard outputs perturbed with Gaussian noise of an equivalent magnitude.

To construct the outputs for the equivalent Gaussian baseline, we match the GMO perturbation amplitudes on a per-sample basis. For a given standard output and its corresponding GMO, we compute the root mean square (γ) of their difference to measure the exact perturbation amplitude. We then draw isotropic Gaussian noise scaled by γ and add it to the standard output. This ensures that the Gaussian-perturbed outputs have the exact same total perturbation energy as the GMOs, but lack the directional guidance derived from the input-output Jacobian.

In Figure 4, we see that GMOs with α equal to 0.05 provide the best-performing generative model and outperform the Gaussian noise baseline.

Qualitatively, with the third panel of Figure 4, we see that the perturbations of GMOs are more similar to the standard output, whereas Gaussian noise seems to create large distortions in the output.[2]

4.2 The Effect of GMOs on FID, Precision, Recall, Density and Coverage

Here, we investigate the underlying mechanics of how GMOs improve generative model performance by examining FID, precision, recall, density, and coverage (Heusel et al., 2017; Kynkäänniemi et al., 2019; Naeem et al., 2020). We evaluate an IMM one-step generator (Zhou et al., 2025) trained on ImageNet256 (Krizhevsky et al., 2012) across a range of GMO coefficients α. For each evaluated curve, we report the optimum performance achieved across a fine-tuning budget $\mathcal{B}$ and merge weight w. Alongside precision and recall, we report density and coverage, their outlier-robust refinements (Naeem et al., 2020). Recall is inflated by a few generated outliers with large neighborhoods, and precision by outliers in the real data.

As illustrated in the left panel of Figure 5, applying GMOs yields a net improvement in FID compared to standard Neon outputs. The FID score drops from its baseline (α equal to zero) to reach an optimal minimum around α equal to 0.2 before slightly increasing at higher perturbation strengths. The precision, recall, density, and coverage as a function of the GMO coefficient provides better insights into how this improvement is achieved. The center and right panels show that as α increases from 0 to 0.4, the model's precision steadily increases while its recall steadily decreases – a pattern that, taken at face value, would suggest GMOs trade diversity for quality. The outlier-robust measures tell a different story: across all architectures (Table 4), density and coverage are maintained or improved from standard-outputs Neon to GMOs, and where the recall estimate dips under GMOs, most visibly on IMM ($-.007$ relative to standard-outputs Neon), coverage instead rises ($+.021$). The FID improvements of GMOs therefore reflect higher robust fidelity together with preserved or improved robust diversity. GMOs improve image quality and diversity simultaneously, rather than trading one for the other.

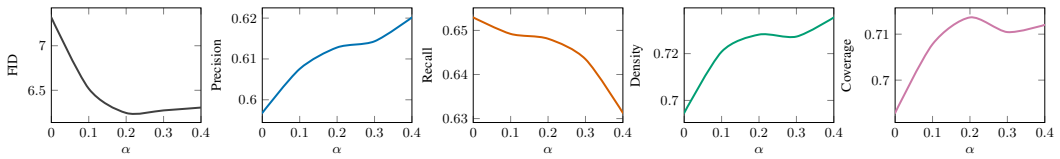


Figure 5: **Neon with GMOs improves image quality and diversity.** For the IMM 1-step generator (Zhou et al., 2025) trained on ImageNet256 (Krizhevsky et al., 2012), we plot FID, precision, recall, density, and coverage (Heusel et al., 2017; Kynkäänniemi et al., 2019) as a function of the GMO coefficient α. Here, α equal to zero corresponds to Neon (Alemohammad et al., 2026) applied to standard outputs, while $\alpha > 0$ corresponds to Neon applied to α-GMOs. Each curve reports the per-α optimum over fine-tuning budget $\mathcal{B}$ and merge weight w. For more details, refer to Appendix D.2.

4.3 GMOs Transfer Across Model Sizes and Inference Strategies

Here, we explore whether GMOs can be transferred across model sizes and inference strategies. In the left panel of Figure 6, we explore the former with MeanFlow models (Geng et al., 2025) trained on ImageNet256 (Krizhevsky et al., 2012). More specifically, GMOs computed from a SiT-B/2 can be successfully used to improve the performance of Neon applied to a larger SiT-L/2 model. Although it should be noted that using GMOs from the SiT-B/2 model to improve the SiT-L/2 is more challenging, as the FID becomes more sensitive to the merging parameter w.

Similarly, in the right panel of Figure 6, we show that GMOs extracted from a one-step IMM (Zhou et al., 2025) generative model can be used to improve a two-step IMM on CIFAR10 (Krizhevsky & Hinton, 2009). These results are significant because computing GMOs for larger generative models or those with more demanding inference strategies is more computationally expensive.

[2]The data space in the MeanFlow is the latent space of a VAE tokenizer (Rombach et al., 2022).

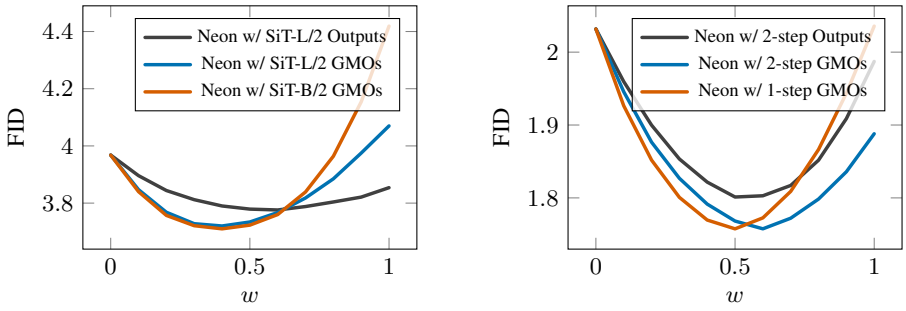


Figure 6: **GMOs are transferable across generative models of different sizes and inference strategies.** In the left panel, we consider using ImageNet256 GMOs from a SiT-B/2 MeanFlow model to improve a SiT-L/2 MeanFlow model using Neon. We compare this to Neon applied directly to the SiT-L/2 model's outputs and GMOs. A computational fine tuning budget of 1.2×10^6 is used in every case, and GMOs are generated with α equal to 0.1. In the right panel, we consider using CIFAR10 GMOs from a one-step IMM model to improve a 2-step IMM model using Neon. We compare this to Neon applied directly to the two-step IMM model's outputs and GMOs. For more experimental detail on this particular experiment, refer to Appendix D.1.

4.4 GMOs Improve State-of-the-Art Generative Models

Here, we demonstrate that Neon with GMOs can improve the performance – measured by FID (Heusel et al., 2017) – more effectively than Neon with standard outputs.

As detailed in Table 1, we evaluate the impact of integrating GMOs into the Neon self-training algorithm across several one-step generative models, including IMM (Zhou et al., 2025), MeanFlow (Geng et al., 2025), and AlphaFlow (Zhang et al., 2026) architectures. The evaluations are conducted on models pre-trained on ImageNet256 (Krizhevsky et al., 2012). The results clearly indicate that self-training with GMOs consistently yields superior generative performance compared to self-training with standard outputs. Across all evaluated model scales, the application of GMOs pushes the FID below both the pre-trained base model baseline and the standard Neon baseline. As shown in Table 1, Neon with GMOs achieves a lower FID than Neon with standard outputs across all five evaluated architectures. Measured relative to the FID reduction that standard Neon achieves over the base model, GMOs increase this reduction by 17–105% across the five architectures, nearly doubling it on IMM and more than doubling it on AlphaFlow SiT-B/2 (see Table 5).

Furthermore, the strength of the negative signal is visible in the finetuning dynamics. Models can attain their optimal FID using GMOs with a smaller fine-tuning budget ($\mathcal{B}$) than is required to reach optimal performance with standard outputs. We quantify this across a broader sweep of α values on a Meanflow model trained on CIFAR10; we find that increasing α systematically lowers both the optimal fine tuning budget $\mathcal{B}$ and the optimal merge weight w required by Neon (Appendix C). This confirms that the targeted signal provided by GMOs not only raises the ultimate performance ceiling but also reaches the optimal checkpoint sooner, because GMOs induce the mode-seeking degradation faster. We show that GMOs statistically significantly improve generative model self-training across all architectures tested (see Table 5.)

4.5 GMOs Improve Negative Guidance Beyond Neon

GMOs operate purely in data space and are agnostic to the self-training algorithm that utilizes them. We chose Neon as the primary algorithm because it is the strongest and most efficient negative guidance method, requiring no additional model at inference and no sampling overhead. To show that our contribution is not solely tied to Neon, we evaluate a second, structurally different algorithm: SIMS-style guidance (Alemohammad et al., 2024b) keeps the finetuned model in the sampling loop and extrapolates predictions at inference time, computing $(1+\omega)\, f_\theta - \omega\, f_{\tilde{\theta}}$ at every sampling step instead of merging weights once.

On IMM, guidance with the model finetuned on standard outputs improves the base FID by 0.75 (from $8.34 \pm .06$ to $7.59 \pm .06$), while the same guidance with the model finetuned on GMOs

Table 1: **The utilization of GMOs in the Neon self-training algorithm yields better generative models.** Here, we implement the Neon self-training algorithm on various one-step generative models trained on ImageNet256. We consider Neon on the standard output of these generators, or on the corresponding GMOs, across various compute levels (as a percentage of pre-training compute), weight-merging parameters w, and α values for GMOs (refer to Appendix D.2 for more details). Among the best-performing configurations, we report the average and standard deviation of the FID across five random seeds and $50,000$ output samples. We provide a repository here containing the model checkpoints obtained using GMOs.

Architecture	Base FID	**Neon w/ Standard Outputs**			**Neon w/ GMOs**			
		$\mathcal{B}$	w	FID	$\mathcal{B}$	w	α	FID
IMM	8.34	$2.5 \times 10^6 (6.1 \times 10^{-3}\%)$	1.6	7.32($\pm$0.07)	$2.87 \times 10^6 (7.0 \times 10^{-3}\%)$	1.6	0.2	$\mathbf{6.25}(\pm\mathbf{0.03})$
MeanFlow SiT-B/2	6.08	$7.2 \times 10^5 (0.25\%)$	0.8	5.70($\pm$0.01)	4.8×10^5 (0.17%)	1.2	0.05	$\mathbf{5.60}(\pm\mathbf{0.01})$
MeanFlow SiT-L/2	3.97	$1.8 \times 10^6 (0.63\%)$	0.5	3.74($\pm$0.02)	$1.8 \times 10^6 (0.63\%)$	0.3	0.1	$\mathbf{3.70}(\pm\mathbf{0.02})$
AlphaFlow SiT-B/2	5.55	$4.8 \times 10^5 (0.16\%)$	0.6	5.36($\pm$0.02)	$4.8 \times 10^5 (0.16\%)$	0.8	0.1	$\mathbf{5.16}(\pm\mathbf{0.02})$
AlphaFlow SiT-XL/2	2.93	$3 \times 10^5 (0.1\%)$	1.2	2.64($\pm$0.02)	$3 \times 10^5 (0.1\%)$	1.1	0.05	$\mathbf{2.59}(\pm\mathbf{0.02})$

improves it by 1.42 (to $6.92 \pm .05$), nearly twice the improvement. GMOs therefore strengthen both consumers of negative signals that we test, weight-space extrapolation and output-space guidance.

5 Discussion

Summary. In this paper, we introduced Geometrically Modified Outputs (GMOs) and demonstrated that they improve the performance of the Neon self-training algorithm for one-step generative models. GMOs are an augmentation technique that modifies model outputs by reweighting the singular values of the generator's input-output Jacobian. This structural augmentation provides a stronger and more targeted negative signal for self-training algorithms, such as Neon and SIMS. Through empirical evaluations, we demonstrated that integrating GMOs into multiple self-training frameworks consistently yields superior generative performance.

Relation to other self-improvement methods. GMOs should be viewed as an enhancement to negative-signal construction, rather than a competing standalone self-training framework. They modify the synthetic outputs used to train the negative model while retaining the downstream fine-tuning and guidance procedures. We demonstrate their utility with both Neon's weight-space extrapolation and SIMS-style inference-time guidance.

The computational requirements of the downstream method remain important. SIMS (Alemohammad et al., 2024b) and Autoguidance (Karras et al., 2024) require auxiliary-model predictions during sampling, while DDO (Zheng et al., 2025) uses multiple self-play rounds and reports a training budget of approximately 12% of pretraining compute. We primarily evaluate GMOs with Neon (Alemohammad et al., 2026), whose merged model requires no additional inference-time evaluations. In this setting, GMOs add offline sample-construction cost while preserving Neon's sampling efficiency. Their effectiveness under both Neon and SIMS motivates investigating whether geometry-based output augmentation can also strengthen other negative-signal methods, including DDO and Autoguidance.

Limitations and Future Directions. Our results suggest that Geometrically Modified Outputs (GMOs) can improve self-training by leveraging the generator's geometric structure, although several directions remain open for future work.

First, GMOs require specifying α. Although outputs for multiple α values can be generated in parallel, applying Neon to each is costly. In our experiments, a fixed $\alpha = 0.1$ without per-model tuning outperforms standard-outputs Neon on four of five ImageNet-trained architectures and is within seed noise on the fifth. Generating GMOs at additional α values is nearly free because they reuse the same probe computation (Algorithm 2); only evaluation and finetuning scale with the number of values tested. However, a search-free heuristic for estimating the optimal α for a given model and dataset would be valuable. Second, we explore GMOs only in the image domain; extending them to other modalities and architectures remains future work.

AI USE STATEMENT

In this work, we used generative AI tools for writing and debugging code for experiments, generating plotting code for figures, and polishing the writing of the manuscript, including proofreading for typographical and grammatical errors, improving clarity and phrasing, and checking consistency of notation, terminology, and cross-references. We have not used generative AI tools for generating research ideas or interpreting results, and the remaining required disclosure tasks are not applicable to this work. We have reviewed all AI-assisted work. In particular, AI-assisted code was reviewed and tested by the authors, all reported numerical results were produced by running this code and checked for consistency across the paper, figures were checked against the underlying data, and every suggested writing edit was manually reviewed against the original text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Epoch AI. Trends in Artificial Intelligence, February 2026. URL https://epoch.ai/trends.

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard Baraniuk. Self-Consuming Generative Models Go MAD. In *International Conference on Learning Representations*, 2024a.

Sina Alemohammad, Ahmed Imtiaz Humayun, Shruti Agarwal, John Collomosse, and Richard Baraniuk. Self-Improving Diffusion Models With Synthetic Data. *arXiv:2408.16333*, 2024b.

Sina Alemohammad, Zhangyang Wang, and Richard Baraniuk. Neon: Negative Extrapolation From Self-training Improves Image Generation. In *International Conference on Learning Representations*, 2026.

Patrick Batsell, Thomas Walker, and Richard Baraniuk. A Geometric Perspective on Recursive Synthetic Training. In *ICLR Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*, 2026.

Mingyang Deng, He Li, Tianhong Li, Yilun Du, and Kaiming He. Generative Modeling Via Drifting. *arXiv:2602.04770*, 2026.

Yunzhen Feng, Elvis Dohmatob, Pu Yang, Francois Charton, and Julia Kempe. Beyond Model Collapse: Scaling up With Synthesized Data Requires Verification. In *International Conference on Learning Representations*, 2025.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One Step Diffusion Via Shortcut Models. In *International Conference on Learning Representations*, 2025.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean Flows For One-step Generative Modeling. In *Advances in Neural Information Processing Systems*, 2025.

Ruifei He, Shuyang Sun, Xin Yu, Chuhui Xue, Wenqing Zhang, Philip Torr, Song Bai, and Xiaojuan Qi. Is Synthetic Data From Generative Models Ready For Image Recognition? In *International Conference on Learning Representations*, 2023.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling Laws for Autoregressive Generative Modeling. *arXiv:2010.14701*, 2020.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-scale Update Rule Converge to a Local Nash Equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, 2017.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arXiv:2001.08361*, 2020.

Tero Karras, Miika Aittala, Tuomas Kynkäänniemi, Jaakko Lehtinen, Timo Aila, and Samuli Laine. Guiding a Diffusion Model With a Bad Version of Itself. In *Advances in Neural Information Processing Systems*, 2024.

Dongjun Kim, Yeongmin Kim, Se Jung Kwon, Wanmo Kang, and Il-Chul Moon. Refining Generative Process With Discriminator Guidance in Score-based Diffusion Models. *arXiv:2211.17091*, 2023.

Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, 2012.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved Precision and Recall Metric for Assessing Generative Models. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. SiT: Exploring Flow and Diffusion-based Generative Models With Scalable Interpolant Transformers. In *European Conference on Computer Vision*. Springer, 2024.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling Data-Constrained Language Models. *arXiv:2305.16264*, 2025.

Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. Reliable fidelity and diversity metrics for generative models. In *International Conference on Machine Learning*, 2020.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models From Natural Language Supervision. In *International Conference on Machine Learning*, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution Image Synthesis With Latent Diffusion Models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer, 2015.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. AI Models Collapse When Trained on Recursively Generated Data. *Nature*, 631(8022), 2024.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-play. *Science*, 362(6419), 2018.

Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. 2019.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency Models. *arXiv:2303.01469*, 2023.

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data Using t-SNE. *Journal of Machine Learning Research*, 9(86), 2008.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will We Run Out of Data? Limits of LLM Scaling Based on Human-generated Data, 2024.

Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. Better Diffusion Models Further Improve Adversarial Training. In *International Conference on Machine Learning*. PMLR, 2023.

Jiawei Yang, Zhengyang Geng, Xuan Ju, Yonglong Tian, and Yue Wang. Representation Fréchet Loss for Visual Generation. *arXiv:2604.28190*, 2026.

Huizhuo Yuan, Zixiang Chen, Kaixuan Ji, and Quanquan Gu. Self-play Fine-tuning of Diffusion Models For Text-to-image Generation. In *Advances in Neural Information Processing Systems*. Curran Associates Inc., 2024.

Kaiyu Yue, Menglin Jia, Ji Hou, and Tom Goldstein. Image Generation With a Sphere Encoder. *arXiv:2602.15030*, 2026.

Huijie Zhang, Aliaksandr Siarohin, Willi Menapace, Michael Vasilkovsky, Sergey Tulyakov, Qing Qu, and Ivan Skorokhodov. AlphaFlow: Understanding and Improving MeanFlow Models. In *International Conference on Learning Representations*, 2026.

Kaiwen Zheng, Yongxin Chen, Huayu Chen, Guande He, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Direct Discriminative Optimization: Your Likelihood-based Visual Generative Model Is Secretly a GAN Discriminator. In *International Conference on Machine Learning*, 2025.

Linqi Zhou, Stefano Ermon, and Jiaming Song. Inductive Moment Matching. In *International Conference on Machine Learning*, 2025.

A PROOF OF THEOREM 1

The standard output of the generator can be decomposed as $\boldsymbol{s} = \boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z} + \boldsymbol{b}_{\boldsymbol{z}}$. Using a singular value decomposition, the Jacobian-vector product $\boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z}$ can be expanded as a sum of its singular components. Let r be the rank of $\boldsymbol{J}_{\boldsymbol{z}}$, then

$$\boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z} = \sum_{k=1}^{r} \sigma_{\boldsymbol{z}}^{(k)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(k)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(k)}$$

Then, by construction, we have

$$\begin{aligned}\tilde{\boldsymbol{J}}_{\boldsymbol{z}}\boldsymbol{z} &= \tilde{\sigma}_{\boldsymbol{z}}^{(1)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)} + \sum_{k=2}^{r} \tilde{\sigma}_{\boldsymbol{z}}^{(k)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(k)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(k)} \\ &= \tilde{\sigma}_{\boldsymbol{z}}^{(1)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)} + \sum_{k=2}^{r} \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(k)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(k)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(k)}\end{aligned}$$

Then, since

$$\sum_{k=2}^{r} \sigma_{\boldsymbol{z}}^{(k)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(k)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(k)} = \boldsymbol{J}_{\boldsymbol{z}}\boldsymbol{z} - \sigma_{\boldsymbol{z}}^{(1)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)},$$

we can write

$$\sum_{k=2}^{r} \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(k)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(k)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(k)} = \sqrt{1-\alpha}\,(\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{z}}) - \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(1)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)}.$$

Meaning,

$$\begin{aligned}\tilde{\boldsymbol{J}}_{\boldsymbol{z}}\boldsymbol{z} &= \tilde{\sigma}_{\boldsymbol{z}}^{(1)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)} + \sqrt{1-\alpha}\,(\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{z}}) - \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(1)} \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)} \\ &= \sqrt{1-\alpha}\,(\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{z}}) + \left(\tilde{\sigma}_{\boldsymbol{z}}^{(1)} - \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(1)}\right) \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)}\end{aligned}$$

Therefore,

$$\tilde{\boldsymbol{s}} = \tilde{\boldsymbol{J}}_{\boldsymbol{z}}\boldsymbol{z} + \boldsymbol{b}_{\boldsymbol{z}} = \sqrt{1-\alpha}\,(\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{z}}) + \left(\tilde{\sigma}_{\boldsymbol{z}}^{(1)} - \sqrt{1-\alpha}\sigma_{\boldsymbol{z}}^{(1)}\right) \left(\boldsymbol{v}_{\boldsymbol{z}}^{(1)\top}\boldsymbol{z}\right) \boldsymbol{u}_{\boldsymbol{z}}^{(1)} + \boldsymbol{b}_{\boldsymbol{z}}.$$

This is precisely the output generated by Algorithm 2, thus the proof is complete. □

A.1 SPECTRAL ENERGY REDISTRIBUTION

The reweighting in Section 3.1 admits a direct interpretation as a redistribution of spectral energy. For a fixed latent vector $\boldsymbol{z}$, write $\sigma_k = \sigma_{\boldsymbol{z}}^{(k)}$ and $\tilde{\sigma}_k = \tilde{\sigma}_{\boldsymbol{z}}^{(k)}$, and let $E = \|\boldsymbol{J}_{\boldsymbol{z}}\|_F^2 = \sum_{k=1}^{r} \sigma_k^2$. For $\alpha \in [0, 1]$, the prescribed transformation satisfies

$$\tilde{\sigma}_1^2 = \sigma_1^2 + \alpha \sum_{k=2}^{r} \sigma_k^2, \qquad \tilde{\sigma}_k^2 = (1-\alpha)\sigma_k^2 \quad (k \geq 2).$$

Thus, the leading component receives exactly the spectral energy removed from the trailing components. Consequently,

$$\|\tilde{\boldsymbol{J}}_{\boldsymbol{z}}\|_F^2 = \tilde{\sigma}_1^2 + \sum_{k=2}^{r} \tilde{\sigma}_k^2 = \|\boldsymbol{J}_{\boldsymbol{z}}\|_F^2.$$

When $E > 0$, define the normalized spectral energies $p_k = \sigma_k^2/E$ and $\tilde{p}_k = \tilde{\sigma}_k^2/E$. The transformation can then be written as

$$\tilde{\mathbf{p}} = (1-\alpha)\mathbf{p} + \alpha\mathbf{e}_1,$$

where $\mathbf{e}_1 = (1, 0, \ldots, 0)^\top$. Hence, α controls a linear interpolation in normalized squared singular values between the original spectrum and one with all spectral energy in the leading component.

The singular directions and affine offset $\boldsymbol{b}_{\boldsymbol{z}}$ are retained. At $\alpha = 0$, the original output is recovered. At $\alpha = 1$, the modified linear term has rank at most one, while the offset remains unchanged. The

preserved quantity is the squared Frobenius norm of the constructed matrix, not necessarily the norm of the generated output.

This provides a design rationale for the GMO reweighting: it introduces a controlled concentration of the local spectrum, motivated by the spectral concentration observed during self-training, while preserving total spectral energy.

B GMOs are Mode Seeking

In Alemohammad et al. (2026), Neon is shown to theoretically work when finetuning against model outputs that are "mode-seeking". Informally, this means that the model outputs are concentrated in high-density regions of the learned data distribution. With Figure 7, we show that GMOs amplify this effect. In the top row of Figure 7, we visualize the CLIP embeddings (Radford et al., 2021) of GMOs across a range of α values using t-SNE. It can be seen that the embeddings become increasingly concentrated for larger values of α, indicating that the images are focusing on the same regions of the data distribution.

With the bottom row of Figure 7, we visualize GMOs for each corresponding value of α. Qualitatively, we observe that the perturbations concentrate on distortions in the output. We provide more examples in Figure 14. In particular, the bottom row of Figure 14 shows that when the standard output has no clear distortions, the corresponding GMO looks relatively unchanged. This supports the idea that GMOs amplify the negative signal in model outputs.

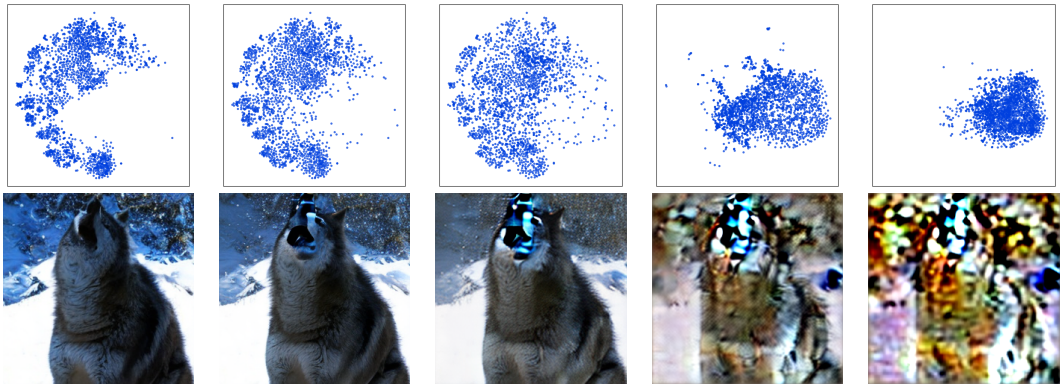


Figure 7: **GMOs amplify the "mode-seeking" nature of model outputs and their errors.** Here we consider a collection of 2048 GMOs for α values 0.0, 0.1, 0.2, 0.6, and 1.0 generated using AlphaFlow (Zhang et al., 2026) SiT-XL/2 (Ma et al., 2024) model trained on ImageNet256 (Krizhevsky et al., 2012). In the top row, we visualize a t-SNE projection (van der Maaten & Hinton, 2008) of the CLIP embeddings (Radford et al., 2021) of the GMOs. In the bottom row, we visualize the GMOs for a specific latent vector.

C The Effect of GMOs on the Compute and Weight-merging of Neon

The performance of the Neon self-training algorithm is dependent on the amount of compute used for finetuning $\mathcal{B}$ and the weight-merging parameter w. In Figure 8, we explore how GMOs affect these hyperparameters for a UNet generative model (Ronneberger et al., 2015) training on CIFAR10 (Krizhevsky & Hinton, 2009) using the MeanFlow framework (Geng et al., 2025). We observe that as the value of α increases, the optimal values for $\mathcal{B}$ and w decrease. This supports the claim that GMOs improve Neon's effectiveness: the more directed degradation induced by GMOs means the optimal checkpoint is reached with less finetuning.

To quantify the complete cost of the pipeline, we follow Neon's analysis (Alemohammad et al., 2026) and report the total cost as a percentage of the model's pretraining compute. Unlike the finetuning

budgets in Table 1, the totals in Table 2 also include the upfront cost of GMO generation. Across all models, the complete pipeline costs between 0.01% and 1.1% of pretraining compute.

Table 2: **Total pipeline cost as a percentage of pretraining compute, including the upfront cost of GMO generation.**

Model	Standard Neon (% of pretraining)	Neon+GMOs (% of pretraining)
IMM	0.007%	0.01%
AlphaFlow-XL/2	0.10%	0.51%
AlphaFlow-B/2	0.16%	0.57%
MeanFlow-B/2	0.26%	0.61%
MeanFlow-L/2	0.63%	1.06%

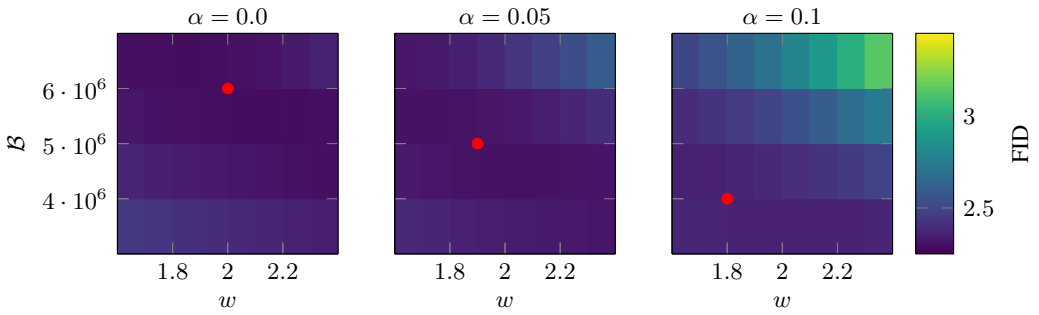


Figure 8: **GMOs can reduce the finetuning required to achieve optimal results using Neon.** Here we apply Neon to a UNet generative model (Ronneberger et al., 2015) training on CIFAR10 (Krizhevsky & Hinton, 2009) using the MeanFlow framework (Geng et al., 2025). We monitor the value of FID for a range of compute levels $\mathcal{B}$ and weight-merging parameters w. With red markers, we indicate which combination of these hyperparameters yields the model with the lowest FID score.

D Experimental Details

D.1 Figure 6

To obtain the result of the right panel of Figure 6, we generate $10,000$ GMOs from 1-step and 2-step IMM (Zhou et al., 2025) models trained on CIFAR10 (Krizhevsky & Hinton, 2009) at an α value of 0.1. This process takes approximately 6 hours using 8 NVIDIA A100-SXM4-80GB GPUs. The 2-step base model is then finetuned using a computational budget of 2.56×10^5. For reference, a computational budget $16,000$ times larger was used to pre-train the base checkpoint (Zhou et al., 2025).

Throughout the finetuning we take 8 checkpoints and evaluate each using weight-merging parameters in the range $[0, 1]$ along a 0.1-step interval. In the right panel of Figure 6, for each finetuning dataset, we visualize the evaluation curves that yielded the lowest FID scores. When using standard outputs from the 2-step model, the optimal FID was seen at a computational budget of 1.6×10^5. When using GMOs from the 2-step model, the optimal FID was seen at a computational budget of 9.6×10^4. When using GMOs from the 1-step model, the optimal FID was seen at a computational budget of 1.28×10^5. For completeness, in Figure 9, we provide the evaluation curves for finetuning checkpoint.

D.2 Table 1

Here we detail the hyperparameter sweeps that were used to obtain the results of Table 1. The following computation times are based on 8 NVIDIA RTX A6000 GPUs.

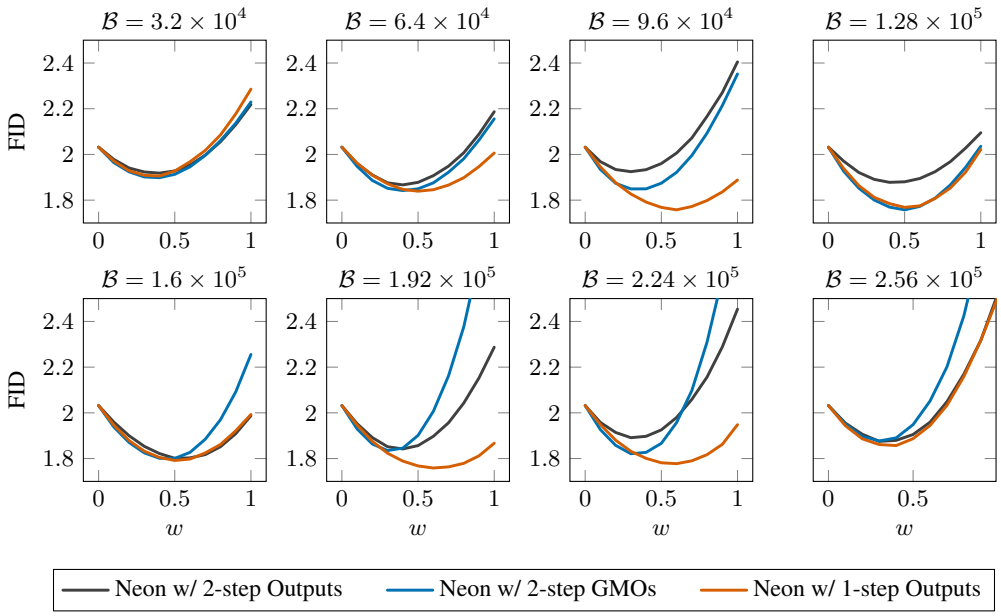


Figure 9: The complete set of evaluation curves for the experiment shown in the right panel of Figure 6, and described in Appendix D.1.

- **MeanFlow SiT-B/2:** We generate $30,000$ standard outputs and GMOs for α values 0.05 and 0.1. This takes approximately 4 hours. The base model is finetuned for 48 epochs on each of these data sets, and we evaluate 6 equally spaced checkpoints. Each round of finetuning takes 1 hour. For each checkpoint, we evaluate the FID using $50,000$ samples and w values ranging from 0.0 to 1.5 at 0.1-step increments. Each FID computation takes approximately 2 minutes.

- **MeanFlow SiT-L/2:** We generate $30,000$ standard outputs and GMOs for α values 0.1 and 0.2. This takes approximately 7 hours. The base model is finetuned for 60 epochs on each of these data sets, and we evaluate 6 equally spaced checkpoints. Each round of finetuning takes 3 hours. For each checkpoint, we evaluate the FID using $50,000$ samples and w values ranging from 0.0 to 1.5 at 0.1-step increments. Each FID computation takes approximately 3 minutes.

- **AlphaFlow SiT-B/2:** We generate $30,000$ standard outputs and GMOs for α values 0.1 and 0.2. This takes approximately 4 hours. The base model is finetuned for 32 epochs on each of these data sets, and we take checkpoints every 4 epochs. Each round of finetuning takes less than 1 hour. For each checkpoint, we evaluate the FID using $50,000$ samples and w values ranging from 0.0 to 1.5 at 0.1-step increments. Each FID computation takes approximately 2 minutes.

- **AlphaFlow SiT-XL/2:** We generate $30,000$ standard outputs and GMOs for α values 0.05 and 0.1. This takes approximately 10 hours. The base model is finetuned for 15 epochs on each of these data sets, and we take checkpoints every 5 epochs. Each round of finetuning takes less than 1 hour. For each checkpoint, we evaluate the FID using $50,000$ samples and w values ranging from 0.0 to 1.6 at 0.1-step increments. Each FID computation takes approximately 4 minutes.

- **IMM (DiT-XL/2):** We generate $30{,}000$ standard outputs and GMOs for α values of 0.1 and 0.2. This takes approximately 5 hours. The base model is finetuned for 109 epochs on each dataset, and we evaluate 8 equally spaced checkpoints. Each round of finetuning takes approximately 5 hours. For each checkpoint, we evaluate FID using $50{,}000$ samples and w values ranging from 0.0 to 1.8 in increments of 0.2. Each FID computation takes approximately 10 minutes.

In Figure 10, we consider each individual model as a row, and show in the left panel what the minimum FID value is for the sweep over w at each compute level. In the right panel, we show the value of w for which the minimum FID was obtained at each compute level.

D.3 SIMS-STYLE GUIDANCE

Here we detail the experiments in Section 4.5. The following computation times are based on 8 NVIDIA RTX A6000 GPUs.

- **IMM (DiT-XL/2):** We use synthetic datasets containing $50{,}000$ standard outputs or GMOs with α values 0.1 and 0.2. For the main sweep, we evaluate 5 checkpoints for standard outputs and 8 checkpoints for GMOs with $\alpha = 0.1$, using guidance strengths $\omega \in \{0.5, 1.0, 1.6\}$. The auxiliary models used for the reported comparison are finetuned for approximately 31 epochs on their respective datasets, requiring approximately 2.5 hours per model. Each FID evaluation uses $50{,}000$ samples, one-step sampling, and classifier-free guidance scale 1.5, and takes approximately 6 minutes. We additionally evaluate $\alpha = 0.2$ using $\omega \in \{0.5, 1.0, 1.6, 2.4\}$; its lowest observed FID is with GMOs at 7.535

Table 3: SIMS-style guidance on IMM ImageNet256. FID is reported as mean $\pm$ sample standard deviation across five evaluations of fixed checkpoints, using $50{,}000$ samples per evaluation. Both guided variants use $\omega = 1.6$.

Method	FID $\downarrow$
Base IMM	8.342 ± 0.058
SIMS, standard outputs	7.591 ± 0.057
SIMS, GMOs ($\alpha = 0.1$)	$\mathbf{6.920 \pm 0.048}$

Table 4 reports precision, recall, density, and coverage for all architectures of Table 1 (five seeds, mean $\pm$ std). Density and coverage, the outlier-robust fidelity and diversity measures (Naeem et al., 2020), are maintained or improved from standard-outputs Neon to GMOs on every architecture, showing that GMOs improve image quality and diversity simultaneously rather than trading one for the other (see Section 4.2). Table 5 reports the corresponding significance analysis for the FID results of Table 1.

Table 4: **Precision, recall, density, and coverage (5 seeds, mean $\pm$ std).** Density and coverage are maintained or improved from standard-outputs Neon to GMOs on every architecture.

Architecture	Method	Precision	Recall	Density	Coverage
IMM (DiT-XL/2)	Base	$.585 \pm .002$	$.647 \pm .001$	$.652 \pm .001$	$.662 \pm .003$
	Neon-std	$.598 \pm .002$	$.654 \pm .002$	$.695 \pm .001$	$.693 \pm .002$
	GMOs	$.615 \pm .003$	$.647 \pm .002$	$\mathbf{.728 \pm .003}$	$\mathbf{.714 \pm .002}$
MeanFlow SiT-B/2	Base	$.717 \pm .002$	$.452 \pm .002$	$1.081 \pm .006$	$.749 \pm .001$
	Neon-std	$.720 \pm .001$	$.452 \pm .003$	$1.110 \pm .006$	$.755 \pm .001$
	GMOs	$.725 \pm .001$	$.451 \pm .003$	$\mathbf{1.124 \pm .007}$	$\mathbf{.759 \pm .002}$
MeanFlow SiT-L/2	Base	$.752 \pm .001$	$.494 \pm .002$	$1.214 \pm .002$	$.828 \pm .001$
	Neon-std	$.764 \pm .000$	$.486 \pm .002$	$1.264 \pm .002$	$.837 \pm .001$
	GMOs	$.763 \pm .001$	$.483 \pm .002$	$\mathbf{1.267 \pm .002}$	$\mathbf{.839 \pm .001}$
AlphaFlow SiT-B/2	Base	$.748 \pm .001$	$.450 \pm .001$	$1.210 \pm .004$	$.794 \pm .001$
	Neon-std	$.743 \pm .002$	$.454 \pm .003$	$1.197 \pm .003$	$.791 \pm .002$
	GMOs	$.740 \pm .001$	$.455 \pm .003$	$1.197 \pm .003$	$.791 \pm .002$
AlphaFlow SiT-XL/2	Base	$.695 \pm .001$	$.596 \pm .001$	$1.005 \pm .002$	$.811 \pm .001$
	Neon-std	$.701 \pm .001$	$.593 \pm .003$	$1.025 \pm .003$	$.820 \pm .002$
	GMOs	$.705 \pm .000$	$.589 \pm .002$	$\mathbf{1.034 \pm .003}$	$\mathbf{.821 \pm .001}$

Table 5: **Statistical significance of the GMO improvement over standard-outputs Neon** (5 seeds, mean $\pm$ std). The final column reports the additional FID reduction of GMOs as a percentage of the reduction Neon itself achieves over the base model.

Architecture	Base FID	Neon-std FID	GMOs FID	Welch p	GMOs' reduction, % of Neon's
IMM (DiT-XL/2)	8.34	$7.32 \pm .07$	$\mathbf{6.25 \pm .03}$	$\sim 10^{-9}$	96%
MeanFlow SiT-B/2	6.08	$5.70 \pm .01$	$\mathbf{5.60 \pm .01}$	2.6×10^{-7}	26%
MeanFlow SiT-L/2	3.97	$3.74 \pm .02$	$\mathbf{3.70 \pm .02}$	0.013	17%
AlphaFlow SiT-B/2	5.55	$5.36 \pm .02$	$\mathbf{5.16 \pm .02}$	2.6×10^{-7}	105%
AlphaFlow SiT-XL/2	2.93	$2.64 \pm .02$	$\mathbf{2.59 \pm .02}$	0.004	17%

E MODEL LICENSES

IMM model weights are obtained through the imm GitHub repository[3] under CC BY-NC-SA 4.0 License. MeanFlow model weights are obtained through the MeanFlow GitHub repository[4] under the MIT license. AlphaFlow model weights are obtained through the AlphaFlow GitHub repository[5] under the Snap Inc. Non-Commercial license.

[3] https://github.com/lumalabs/imm

[4] https://github.com/zhuyu-cs/MeanFlow

[5] https://github.com/snap-research/alphaflow

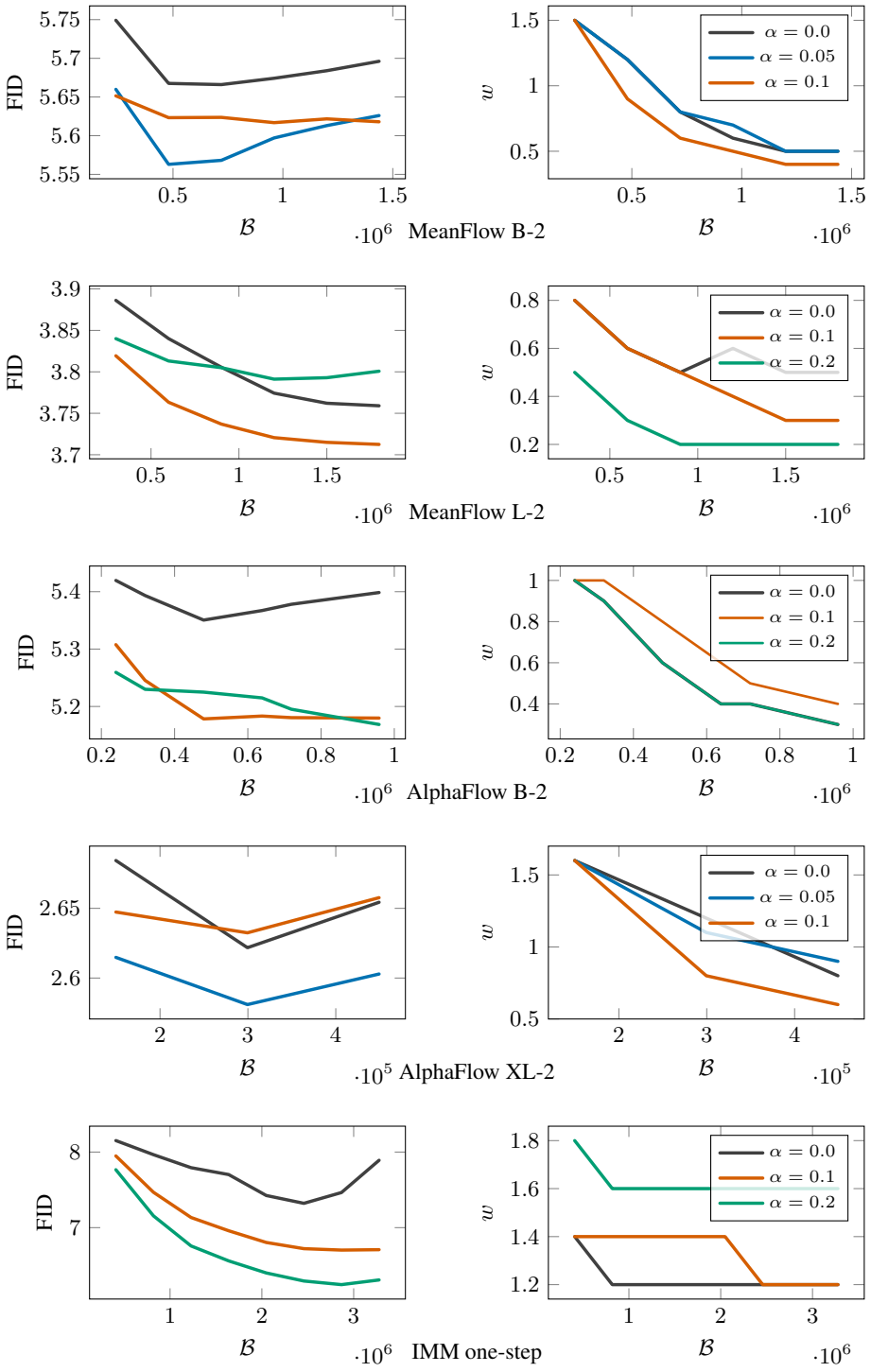


Figure 10: Hyperparameter sweeps described in Appendix D.2 that yielded the results in Table 1.

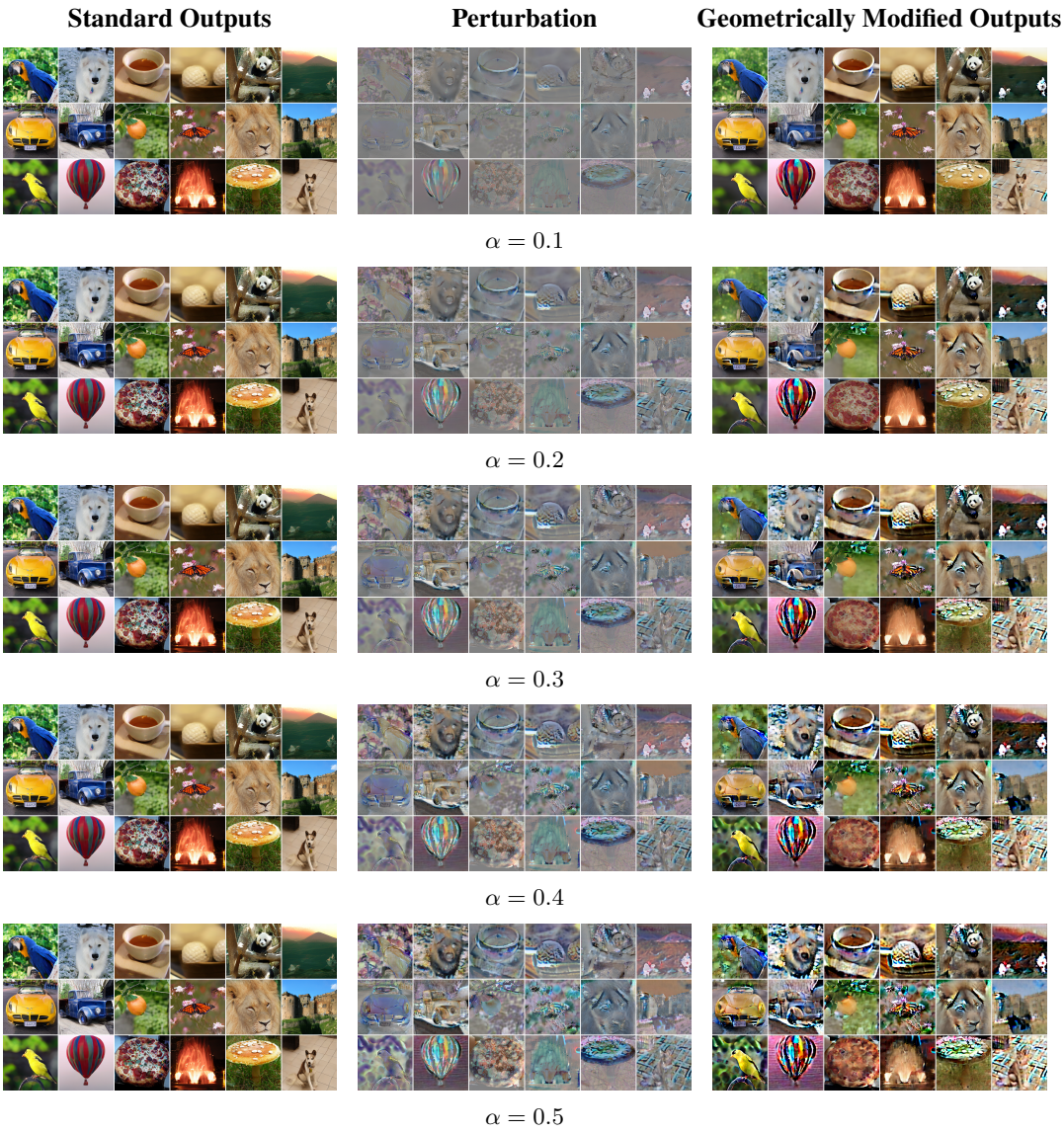


Figure 11: Additional examples of those shown in Figure 1 at different GMO α levels.



Figure 12: Qualitative comparison on IMM ImageNet-256 (Part I).

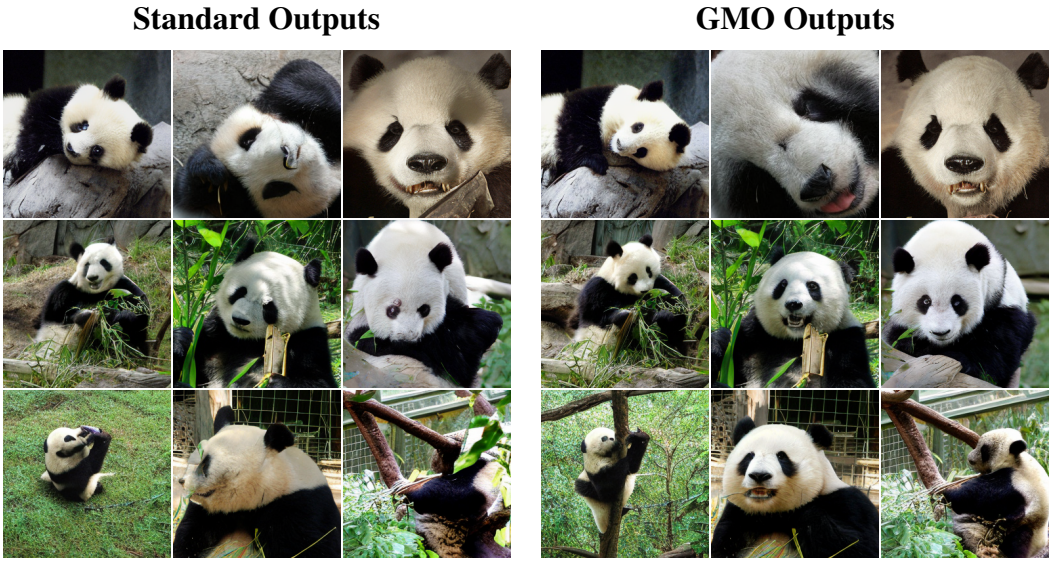


Figure 12: Qualitative comparison on IMM ImageNet-256 (Part II).

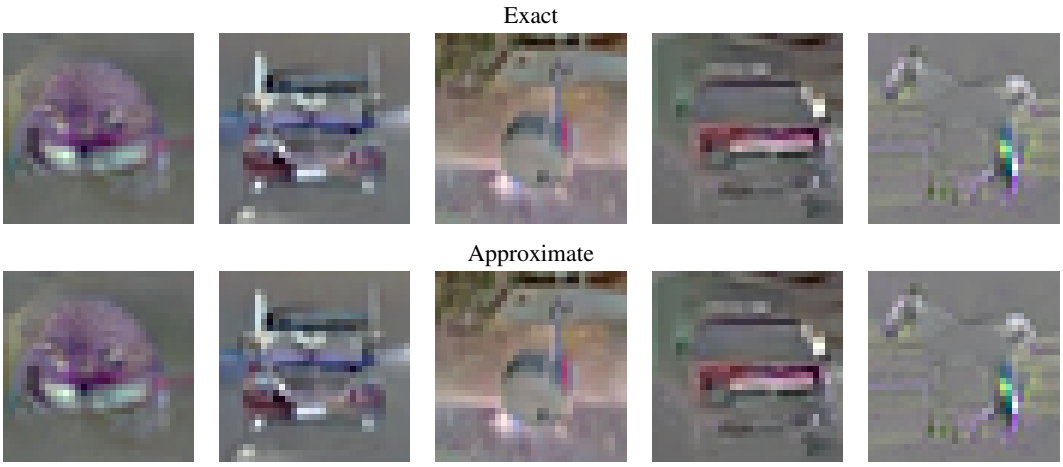


Figure 13: In Figure 3, we quantitatively demonstrate that power iteration can be used to accurately approximate the singular vectors of the input-output Jacobians of generative models. Here, we qualitatively support this by showing the exact (top) and approximate top left singular vectors of these input-output Jacobians.

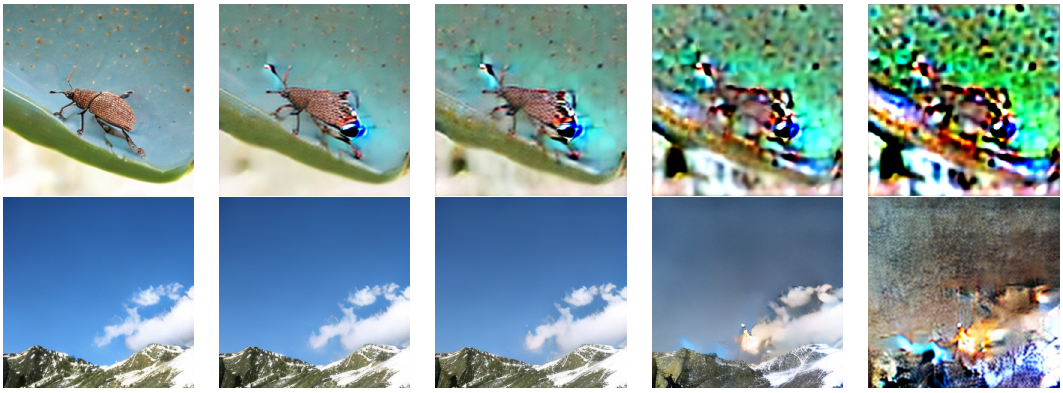


Figure 14: Here we provide additional examples to that illustrated in the bottom row of Figure 7.

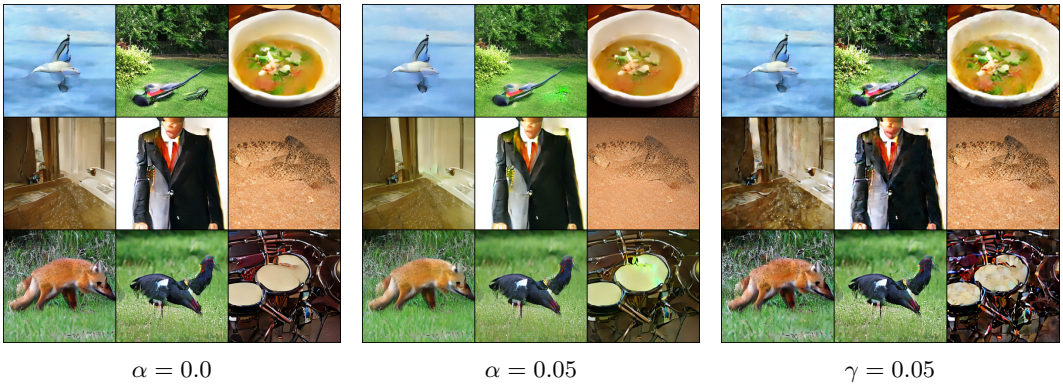


Figure 15: Here we provide additional examples to complement the third panel of Figure 4.