DriftWeight: Repulsive Drift in Mean-Flow Space for Neural Network Weight Generation

Reo Iizuka
Boston Consulting Group

Hiroaki Shioya
Boston Consulting Group

Naoto Yokoya
The University of Tokyo
RIKEN AIP

Abstract

Generating neural network weights in a single forward pass promises to enable fast model sampling for transfer and model-set construction. Yet accuracy alone is insufficient: existing generators can map diverse latent inputs to weights that remain functionally close to their training checkpoints. We study this *functional mode collapse* in the one-step regime and introduce DriftWeight, a MeanFlow-based generator with temperature-scaled repulsive drift in a checkpoint-augmented PCA proxy space. The method uses structured negatives from the current batch and optimization history to push generated weights away from observed clusters while preserving task performance. Our key finding is that repulsion is not automatically compatible with one-step inference. In a controlled ablation, applying the drift loss at the same boundary evaluation used for sampling collapses one-step accuracy even though multi-step integration remains accurate. DriftWeight mitigates this boundary-localized failure with a consistency objective that anchors the inference point to the training-weight manifold. Across MNIST, Fashion-MNIST, and CIFAR-10, DriftWeight matches or exceeds reported 100-step DeepWeightFlow accuracy in a single forward pass. On MNIST, where we directly evaluate error-set functional overlap, it reduces MaxIoU from the DeepWeightFlow regime of approximately 0.82 to 0.66. On CIFAR-100 with a ViT-Base target ($\sim$86M parameters), one-step samples remain within 1.7 percentage points of the training-population mean.

1 Introduction

Training a neural network from scratch is expensive. Once a source population and a weight generator have been trained, additional models can be sampled without training each model from scratch. This amortizes the preparation cost over repeated sampling. Such models could support applications such as model-set construction, transfer learning, lightweight model editing, and periodic parameter refresh for deployed systems [Gupta et al., 2026, Erdogan, 2025, Saragih et al., 2025b, Izmailov et al., 2020]. For these applications, however, accuracy alone is not enough: generated models should not repeatedly collapse onto the same functional neighborhoods. We therefore focus on collapse-sensitive functional diversity, measured through error-set overlap, which directly tests whether generated models repeatedly fail on the same examples as observed checkpoints.

This is precisely where current weight generators fall short. Neural network weights are high-dimensional, shaped by strong permutation symmetries [Entezari et al., 2022], and tend to concentrate near thin manifolds [Garipov et al., 2018, Izmailov et al., 2020]. Zeng et al. [2025] showed that several weight generators produce outputs that are little more than replicas or local interpolations of their training set. We call this *functional mode collapse*: generated weights remain accurate, yet exhibit high error-set overlap with observed training checkpoints.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

The problem is especially acute for *one-step* generation. Existing approaches—hypernetworks [Ha et al., 2017, Schürholt et al., 2022], diffusion-based generators [Wang et al., 2024, Soro et al., 2025, Wang et al., 2025], and flow matching methods [Lipman et al., 2023, Gupta et al., 2026]—make meaningful progress on accuracy, while MeanFlow [Geng et al., 2026] pushes generation to a single network evaluation. But in weight space, fast generation can map diverse latent inputs back to a narrow region around the observed checkpoints. Existing methods do not directly address this functional collapse, nor do they explain why naively adding a diversity-promoting objective can fail in the one-step regime.

This paper introduces DriftWeight, a one-step neural network weight generator that augments MeanFlow with temperature-scaled repulsive drift in a checkpoint-augmented PCA space (Figure 1). The method is built around three requirements specific to weight generation. First, repulsion needs a scalable, label-free, task-agnostic proxy space in which distances provide useful but imperfect signal about functional change; we use checkpoint-augmented PCA for this purpose (Section 4.2). Second, negative samples should reflect the optimization trajectories that produced the training weights, so DriftWeight uses current samples, same-seed history, and other-seed history as structured negatives. Third, and most importantly, repulsive drift in a one-step generator must be balanced by an on-manifold anchor at the exact boundary point used for inference.

Our controlled MNIST ablation identifies a *boundary-localized failure mode* that can arise when repulsive drift is applied to a one-step weight generator. The drift loss backpropagates through the single one-step evaluation $u_\theta(z_1, 1, 0)$; without an on-manifold target at this same point, it can push the boundary predictor off the valid-weight manifold even when the interior flow remains well behaved. Empirically, removing consistency while keeping drift collapses one-step accuracy from 96.3% to 10.5%, while 100-step accuracy remains 95.9% (Table 4). This failure mode is invisible under multi-step-only evaluation. DriftWeight resolves it with a MeanFlow-style consistency objective, which supplies an on-manifold target at the boundary evaluation used for one-step inference.

Empirically, DriftWeight matches or exceeds reported 100-step DeepWeightFlow accuracy in a single forward pass on MNIST, Fashion-MNIST, and CIFAR-10. The direct comparative evidence for reduced error-set overlap is established on MNIST, where corresponding baseline MaxIoU results are available: DriftWeight reduces MaxIoU from ≈ 0.82 to ≈ 0.66, moving closer to the independent-training reference. Larger settings provide complementary accuracy, boundary-failure, and proxy-geometry evidence. We also demonstrate generation beyond small models: on CIFAR-100 with a ViT-Base target ($\sim$86M parameters), one-step samples remain within 1.7 percentage points of the training-population mean. We study these claims in image-classification weight generation, where both accuracy and error-set functional overlap can be measured directly.

Contributions.

1. We identify functional mode collapse as a central obstacle for one-step weight generation and demonstrate, in a controlled small-scale ablation, that naive repulsive regularization can fail at the one-step boundary.
2. We propose DriftWeight, which combines checkpoint-augmented PCA, optimization-process-aware negative sampling, and a boundary-level consistency anchor to make repulsive one-step generation work in weight space.
3. We show that DriftWeight improves the accuracy–MaxIoU trade-off on MNIST under a collapse-sensitive functional-overlap metric while preserving the generator-latency advantage of one-step inference.

The code is publicly available at `https://github.com/Reo-I/DriftWeight-NeurIPS2026`.

Figure 1 summarizes how DriftWeight collects trajectory checkpoints, embeds them with history-aware PCA, and trains a one-step MeanFlow generator with repulsive drift and boundary consistency.

2 Related Work

Neural network weight generation. Our work lies within *Weight Space Learning*, which treats neural parameters as a structured data modality [Han et al., 2026, Navon et al., 2023, Shamsian et al., 2024]. Weight generation has evolved from hypernetworks [Ha et al., 2017, Ratzlaff and Li, 2019,

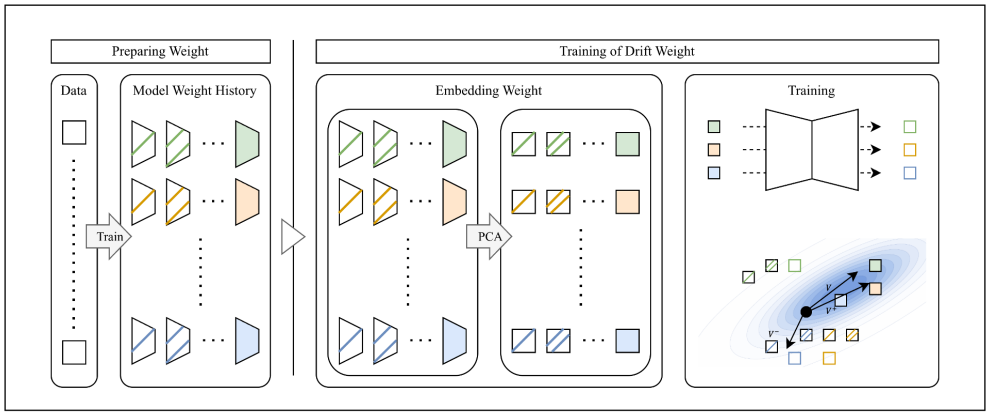


Figure 1: Overview of the DriftWeight pipeline. **(1) Preparing Weight**: K_{PCA} checkpoints are collected along N independent training trajectories (rows = seeds, colors = seed identity). **(2) Embedding Weight**: checkpoints are projected by a PCA basis fitted on the full history, raising the sample-count upper bound on PCA rank from $N-1$ to $NK_{\mathrm{PCA}}-1$. **(3) Training**: a MeanFlow generator u_θ maps noise to weight codes in a single step, trained with a consistency loss plus a temperature-scaled repulsive drift loss whose attractive (V^+) and repulsive (V^-) components produce a drift field V (blue density and arrows) pushing the generator away from known weight clusters.

Knyazev et al., 2021] and hyper-representations [Schürholt et al., 2022, 2024] to direct diffusion- and flow-based generators [Wang et al., 2024, Soro et al., 2025, Wang et al., 2025, Saragih et al., 2025b,a, Erdogan, 2025]. RPG models dependencies among parameter tokens recurrently and uses the resulting representations to condition diffusion-based parameter generation [Wang et al., 2025]. SANE learns representations of weight subsets through sequential processing and supports sequential generation of new models [Schürholt et al., 2024]. The closest prior method is DeepWeightFlow, which combines weight-space flow matching with Git Re-Basin/TransFusion canonicalization and PCA preprocessing for strong multi-step complete-weight generation [Ainsworth et al., 2023, Rinaldi et al., 2025, Gupta et al., 2026]. DriftWeight instead targets the accuracy–MaxIoU trade-off of one-step generation and explicitly addresses functional collapse.

One-step and few-step generation. Reducing generation steps is a major theme across generative modeling: Flow Matching [Lipman et al., 2023] and Rectified Flow [Liu et al., 2023] cast sampling as vector-field regression over probability paths, Consistency Models [Song et al., 2023] learn direct noise-to-data maps, and MeanFlow [Geng et al., 2026] learns average velocity fields for one-step generation. Drifting [Deng et al., 2026] reshapes the pushforward distribution during training with an auxiliary drift field. These methods were developed primarily for image generation; to our knowledge, DriftWeight is the first to combine a MeanFlow-style backbone with drifting-style repulsion in neural weight space, where repulsive objectives can introduce a boundary-specific failure mode.

Weight-space geometry and subspaces. Neural weights are shaped by permutation symmetries and low-loss connectivity [Garipov et al., 2018, Entezari et al., 2022, Lim et al., 2024]. Alignment methods such as Git Re-Basin and TransFusion [Ainsworth et al., 2023, Rinaldi et al., 2025] and model soups [Wortsman et al., 2022] show that compatible weights can be averaged, while Subspace Inference [Izmailov et al., 2020] shows that SGD trajectories span useful low-dimensional regions. DeepWeightFlow fits PCA on N terminal checkpoints, capping the basis at $N-1$ dimensions. DriftWeight instead fits PCA on terminal and intermediate checkpoints, yielding a history-aware coordinate system for repulsive one-step generation.

Memorization and diversity evaluation. Zeng et al. [2025] showed that high downstream accuracy can hide replication or interpolation of training checkpoints, exposing an evaluation gap for weight generators. This is especially relevant for applications such as ensembling and uncertainty estimation, where generated models must provide information beyond a single high-accuracy checkpoint [Lakshminarayanan et al., 2017, Maddox et al., 2019]. DriftWeight adds an explicit repulsive bias against observed clusters; its structured negatives are a weight-space analogue of contrastive memory banks

such as Repulsor [Zhang et al., 2025]. We use MaxIoU as a collapse-sensitive primary metric for error-set functional overlap.

3 Background

Generative flow models in one step. Flow Matching [Lipman et al., 2023] regresses a velocity field $u_\theta(z_t, t)$ along a prescribed path such as $z_t{=}(1{-}t)x_1{+}te$ and generates by integrating the probability-flow ODE; Rectified Flow [Liu et al., 2023] reduces step counts by straightening such paths. MeanFlow [Geng et al., 2026] instead learns the *average* velocity $u_\theta(z_t, r, t)$ over $[r, t]$, so the boundary evaluation $(r, t) = (0, 1)$ gives the one-step map $\hat{x} = z_1 - u_\theta(z_1, 1, 0)$. This makes MeanFlow attractive for fast sampling, but by itself it does not prevent diverse latent inputs from mapping back to a narrow region around observed weights.

Drifting and repulsive regularization. Drifting [Deng et al., 2026] adds an auxiliary field $V(x){=}V^+(x){-}V^-(x)$ and trains the generator toward $\tilde{x} = x + \eta V(x)$; repulsor-style objectives use negative banks for a similar anti-collapse signal [Zhang et al., 2025]. These ideas were developed for image generation, where feature distances are approximately Euclidean and outputs are not subject to permutation symmetries. In weight space, distances depend on the chosen representation (raw vectors, canonicalized weights, or PCA codes), so repulsive training needs both a meaningful coordinate system and negatives reflecting the optimization history.

Functional mode collapse. We formalize the failure mode targeted in this paper.

Definition 1 (Functional mode collapse)**.** *Let $g : \mathcal{Z} \to \mathcal{W}$ be a weight generator with training set $\mathcal{W}_{\mathrm{tr}}$ and let $\mathrm{MaxIoU}(w){=}\max_{w' \in \mathcal{W}_{\mathrm{tr}} \setminus \{w\}} \mathrm{IoU}(E(w), E(w'))$, where $E(w)$ is the set of test inputs on which w errs. We say g exhibits* functional mode collapse *at level τ if $\mathbb{E}_z[\mathrm{MaxIoU}(g(z))] \geq \tau$ while the population accuracy matches $\mathcal{W}_{\mathrm{tr}}$.*

This captures the downstream failure case where generated models are accurate but redundant. Empirically, independently trained classifiers reach leave-one-out MaxIoU ≈ 0.47 (Section 5.4), whereas additive Gaussian noise saturates at ≈ 0.82–0.85; Appendix A relates MaxIoU to predictive-distribution distance.

Why a naive combination can fail. MeanFlow provides speed but not diversity; drifting provides repulsion but needs a meaningful weight-space geometry; and terminal-only PCA compresses weights but is capped at $N{-}1$ components. DriftWeight therefore combines checkpoint-augmented PCA, trajectory-aware negatives, and an on-manifold consistency signal at the one-step boundary. The next section details these three design choices.

4 Method

DriftWeight (Fig. 1) retains the MeanFlow backbone and augments it with (i) a PCA-based learning space (Section 4.2), (ii) a drifting loss with structured positives and negatives (Sections 4.3–4.5), and (iii) one-step consistency. Data collection is described in Section 5.1.

4.1 Problem formulation

Let $\mathcal{W} = \{w_i\}_{i=1}^N$ be a collection of trained weights. We learn a generator that samples new weights $\hat{w} \sim p(w)$ with strong downstream performance while avoiding collapse toward $\mathcal{W}$. Training takes place in a PCA learning space fitted on $\mathcal{W}_{\mathrm{fit}} = \mathcal{W}_{\mathrm{final}} \cup \mathcal{W}_{\mathrm{history}}$ (union of final and history checkpoints), with representation $x = \psi(w) = P(w - \mu)$ where P is the PCA projection and μ the empirical mean; generation, positives, negatives and drift are all computed in this space, and the inverse transform is applied only at evaluation time. The MeanFlow base generator produces one-step samples $\hat{x}_{1\to 0} = z_1 - u_\theta(z_1, 1, 0)$, which DriftWeight feeds into the drift branch.

We also consider an *additional-training* setting motivated by transfer learning. A sampled or source weight vector initializes the target model, $\hat{w}_0$, and a small number of gradient epochs on a new dataset produces $\hat{w}_K = \mathcal{A}_K(\hat{w}_0; \mathcal{D}_{\mathrm{new}})$. This setting tests whether generated weights are useful not only as complete classifiers without finetuning, but also as warm-starts for rapid adaptation.

4.2 Repulsive PCA weight space

PCA plays two roles: it compresses weights and defines the geometry in which distances, affinities, and negative hardness are computed. We use it as a scalable, label-free, task-agnostic proxy space rather than as an optimal function-space embedding. This choice is motivated by Subspace Inference [Izmailov et al., 2020] and by the need to apply repulsion without a task-specific reference set. PCA fitted on N terminal checkpoints yields at most $N-1$ components, capping the available basis at 99 dimensions for $N=100$ [Gupta et al., 2026]. We augment the PCA fitting set with intermediate checkpoints from the same training trajectories, expanding the available subspace without additional source-model training runs. With K_{PCA} total checkpoints per seed, including the terminal checkpoint, the sample-count upper bound on PCA rank becomes $NK_{\mathrm{PCA}}-1$ (e.g. 4,999 at $N=100$, $K_{\mathrm{PCA}}=50$). The retained dimension is a separate choice; checkpoint storage grows linearly with N, K_{PCA}, and model size.

4.3 Drifting loss

Let x_{gen} denote the current generated sample in learning space, let $Y^+ = \{y_j^+\}_{j=1}^{N_+}$ be the positive set, and let $Y^- = \{y_k^-\}_{k=1}^{N_-}$ be the negative set. In the current implementation, Y^+ is the target batch itself, while Y^- is constructed from generated samples and checkpoint history, described in the next subsection.

Within this PCA learning space, we normalize features $\phi(v) = v/\sqrt{\frac{1}{d}\|v\|_2^2 + \varepsilon}$ and build a drift field from temperature-scaled distances between generated samples, positives, and negatives. For each temperature, positives form an attractive barycenter that keeps samples near the training-weight manifold, while structured negatives form a repulsive barycenter that pushes samples away from current and historical clusters. Aggregating the resulting normalized directions across temperatures gives V_{agg}; Appendix B gives the exact affinity normalization and hard-negative selection. The drift loss is

$$\mathcal{L}_{\mathrm{drift}} = \|\phi(x_{\mathrm{gen}}) - \mathrm{sg}(\phi(x_{\mathrm{gen}}) + V_{\mathrm{agg}})\|_2^2, \tag{1}$$

with $x_{\mathrm{gen}} = \hat{x}_{1\mathrm{step}}$ at the one-step boundary. All signals under sg are detached, so the only gradient path into θ is through the single evaluation $u_\theta(z_1, 1, 0)$—a boundary localization central to Section 4.5. The objective does not alter inference, only the training signal.

4.4 Structured negative sampling

The negative set is tailored for weight generation: $Y^- = \mathrm{concat}(Y^-_{\mathrm{current}}, Y^-_{\mathrm{same}}, Y^-_{\mathrm{other}})$, containing current generated samples, same-seed history checkpoints, and other-seed history checkpoints. A fixed negative budget first reserves the current batch and then allocates the remaining history budget between same-seed and other-seed checkpoints. History checkpoints are sampled with weights depending on training progress. The three categories give a clear geometric interpretation: current negatives enforce within-batch diversity, same-seed history prevents returning to earlier points of the same optimization path, and other-seed history discourages collapse toward regions of other training runs.

4.5 One-step consistency as an on-manifold anchor

DriftWeight uses a one-step consistency objective inherited from MeanFlow. Given noise $z_1 \sim \mathcal{N}(0, \sigma_s^2 I)$, the one-step prediction $\hat{x}_{1\mathrm{step}} = z_1 - u_\theta(z_1, 1, 0)$ is regressed onto a stop-gradient two-step teacher $\hat{x}_{2\mathrm{step}}$ obtained by sampling $d \sim \mathcal{U}(0.05, 0.95)$ and stepping $1 \to 1-d \to 0$:

$$\hat{x}_{1\mathrm{step}} = z_1 - u_\theta(z_1, 1, 0), \qquad \mathcal{L}_{\mathrm{cons}} = \|\hat{x}_{1\mathrm{step}} - \mathrm{sg}(\hat{x}_{2\mathrm{step}})\|_2^2. \tag{2}$$

In standard MeanFlow training this term is often optional. In the drifted setting, however, it plays a crucial structural role, because of the following *boundary localization* argument. The drift loss (Eq. 1) is applied to $x_{\mathrm{gen}} = \hat{x}_{1\mathrm{step}}$ with all other signals detached, so its gradient flows back into θ through the *single* network evaluation $u_\theta(z_1, 1, 0)$—exactly the point used at 1-step inference. The MeanFlow base loss, in contrast, distributes its signal over all sampled (t, r), so only a small fraction of batches hits the corner $\{t{=}1, r{=}0\}$. Furthermore, the drift target $x_{\mathrm{gen}} + V_{\mathrm{agg}}$ is an off-manifold repulsive displacement. Consistency (Eq. 2) is the only additional training term that is

also evaluated at $u_\theta(z_1, 1, 0)$ and whose regression target $\hat{x}_{2\text{step}}$ is on-manifold, because the two-step teacher traverses interior times $(t < 1, r > 0)$ that are well-supervised by the base MeanFlow loss. Consistency therefore acts as an *on-manifold anchor* at the 1-step boundary, counteracting the drift gradient's off-manifold tendency. Removing consistency while keeping drift collapses the boundary but leaves the interior velocity field intact (Section 5.5, Table 4).

4.6 Full objective and auto-scaling

Let $\mathcal{L}_{\text{base}}$ denote the MeanFlow base loss and let λ_{cons} be the coefficient of the one-step consistency term. The full training objective is

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{base}} + \lambda_{\text{cons}}\mathcal{L}_{\text{cons}} + \lambda_{\text{drift,eff}}\mathcal{L}_{\text{drift}}. \tag{3}$$

Because the drift magnitude depends on temperature, normalization and pool size, we auto-scale it with a scalar loss-balancing rule based on EMAs of the component losses, in the spirit of adaptive multi-task loss balancing [Chen et al., 2018, Liu et al., 2019]. This preserves the repulsive direction while keeping its scale comparable to the on-manifold MeanFlow and consistency terms; the exact formula and alternatives are in Appendix B.

The full training procedure is summarized in Algorithm 1 (Appendix B).

5 Experiments

We evaluate DriftWeight on three image classification benchmarks and answer three questions: (1) is downstream accuracy competitive with a strong multi-step baseline? (2) does the repulsive objective improve collapse-sensitive functional overlap? and (3) which components contribute most to performance?

5.1 Experimental setup

Datasets and architectures. We follow the DeepWeightFlow evaluation protocol [Gupta et al., 2026]: MNIST with a 3-layer MLP (784-32-32-10, ≈26K params), Fashion-MNIST with a 3-layer MLP (784-128-128-10, ≈118K params), and CIFAR-10 with ResNet-18 (≈11.2M) and ResNet-20 (≈0.27M). These architectures and target-model training recipes match the DeepWeightFlow setup [Gupta et al., 2026, Table 12]. Each training set contains 100 independently trained seeds; intermediate checkpoints from each trajectory are stored as history for PCA fitting and negative sampling.

Baselines. We compare against **DeepWeightFlow** [Gupta et al., 2026] (Flow Matching + Git Re-Basin + PCA, 100-step RK4; numbers from the original paper), **FlowMatching** (100-step, our reimplementation), **MeanFlow** (no drift or consistency loss), and **MeanFlow + PCA** (PCA space, no drift). Our DeepWeightFlow benchmark comparisons follow the published architectures and source-model training recipes, but use independently trained source populations. The accuracy tables report each method's source-population accuracy to make differences in the starting populations explicit.

Evaluation metrics. Over 100 generated models we report: (i) **test accuracy** (mean ± std); (ii) **MaxIoU** of wrong predictions (lower = less error-set overlap; Section 5.4); and (iii) **generation steps**.

Training details. All DriftWeight generators use AdamW with weight decay 10^{-5} and evaluate 100 generated models at one step; multi-step reference results are reported in Appendix D. PCA is fitted either on terminal checkpoints or on the union of terminal and history checkpoints, depending on the experiment, and retains the dimensions reported in Appendix C. These dimensions balance retained variance and generator tractability; they are not claimed to be minimal or optimal. The default drift temperatures are $\mathcal{T} = \{0.02, 0.05, 0.2\}$ with EMA auto-scaling. Appendix C gives the full target-model, PCA, generator, negative-sampling, evaluation, and hardware settings.

Table 1: Comparison with existing weight generation methods for complete MNIST classifier generation, without finetuning. All baseline numbers are reproduced from Gupta et al. [2026]. Each method reports the accuracy of its own training population ("Original") alongside the generated accuracy. DriftWeight generates weights in a **single forward pass**.

Model	Network	Original Acc.	Generated Acc.	Steps	Ref.
DeepWeightFlow (w/ Re-Basin)	3-Layer MLP	$96.32_{\pm 0.20}$	$96.17_{\pm 0.31}$	100	[Gupta et al., 2026]
DeepWeightFlow (w/o Re-Basin)			$96.19_{\pm 0.27}$	100	
WeightFlow	3-Layer MLP	93.3	78.6	–	[Erdogan, 2025]
FLoWN (Unconditioned)	medium-CNN	92.76	83.58	–	[Saragih et al., 2025b]
DriftWeight (ours)	3-Layer MLP	$96.30_{\pm 0.19}$	$\mathbf{96.30}_{\pm 0.14}$	**1**	

Table 2: Comparison with existing weight generation methods for complete ResNet-18 CIFAR-10 classifier generation, without finetuning. All baseline numbers are reproduced from Gupta et al. [2026]. "Partial" denotes methods that generate only a subset of parameters (e.g. excluding batch-norm). ‡Models generate weights through recurrent or sequential processing of parameter subsets. DriftWeight generates complete weights in a **single forward pass**.

Model	Original	Generated (Partial)	Generated (Complete)	Steps	Ref.
DeepWeightFlow (w/ Re-Basin)	$94.45_{\pm 0.14}$	–	$93.55_{\pm 0.13}$	100	[Gupta et al., 2026]
DeepWeightFlow (w/o Re-Basin)		–	$93.47_{\pm 0.20}$	100	
RPG‡	95.3	–	95.1	–	[Wang et al., 2025]
SANE‡	$92.14_{\pm 0.12}$	–	$68.6_{\pm 1.2}$	–	[Schürholt et al., 2024]
D2NWG	94.56	$94.57_{\pm 0.0}$	–	–	[Soro et al., 2025]
$\mathcal{NM}$ (Unconditioned)	94.54	94.36	–	–	[Saragih et al., 2025a]
P-diff (best NN)	94.54	94.36	–	–	[Wang et al., 2024]
FLoWN (best NN)	94.54	94.36	–	–	[Saragih et al., 2025b]
DriftWeight (ours)	$93.44_{\pm 0.22}$	–	$\mathbf{93.64}_{\pm 0.12}$	**1**	

5.2 Main results: generation quality

Tables 1–2 compare DriftWeight against existing weight generators. On MNIST, DriftWeight at one step (96.30%) surpasses DeepWeightFlow's 100-step result (96.17%); WeightFlow and FLoWN trail by a wide margin. On CIFAR-10 ResNet-18, DriftWeight trained on our 100-seed source population ($93.44_{\pm 0.22}\%$) reaches 93.64% in one step, exceeding DeepWeightFlow's 100-step result (93.55%); RPG reaches 95.1% but uses recurrent diffusion and a source population with higher accuracy. Fashion-MNIST (89.87% vs. 89.10%) and ResNet-20 (77.97% vs. 75.07%) confirm the trend; multi-step reference numbers are in Appendix D (Tables 7–8). A CIFAR-100 scale-up with ViT-Base ($\sim$86M params) is reported in Appendix D.7 (Table 13).

5.3 Transfer via additional training

To connect complete weight generation with downstream adaptation, we follow the DeepWeightFlow transfer protocol: CIFAR-10 ResNet-18 weights initialize a classifier that is evaluated on STL-10 and SVHN at epoch 0 and after 1 or 5 fine-tuning epochs. Table 3 reports our generated, CIFAR-10 source, and random initializers under the same implementation, alongside the reported DeepWeightFlow and FLoWN numbers from Gupta et al. [2026] for context. The goal is not to claim a controlled head-to-head transfer comparison, since source populations and implementations differ, but to test whether one-step generated weights preserve the transfer utility of the CIFAR-10 source checkpoints. DriftWeight generated weights perform comparably to independently trained CIFAR-10 initializers after short fine-tuning and substantially outperform random initialization.

Under the matched implementation, DriftWeight generated initializers closely track the independently trained CIFAR-10 source checkpoints on both targets: after five epochs they reach 86.06% vs. 85.47%

Table 3: Additional-training transfer protocol for CIFAR-10 ResNet-18 initializers. We report target test accuracy before target fine-tuning (epoch 0) and the best accuracy reached within 1 or 5 fine-tuning epochs. DeepWeightFlow [Gupta et al., 2026] and FLoWN [Saragih et al., 2025b] rows are reported numbers from Gupta et al. [2026]; DriftWeight, Original CIFAR-10, and RandomInit use our matched implementation with n=5 initializers, CIFAR-10 BatchNorm recalibration for generated weights, native STL-10 label mapping, and AdamW fine-tuning.

Method	STL-10			SVHN		
	Epoch 0	Epoch 1	Epoch 5	Epoch 0	Epoch 1	Epoch 5
DeepWeightFlow	48.32±0.34	79.69±1.08	84.63±0.17	11.57±0.49	91.66±0.79	95.85±0.09
FLoWN	35.16±1.24	36.15±1.14	37.43±1.19	**17.99**±0.82	68.64±7.07	77.36±1.07
DriftWeight (ours)	**48.97**±0.07	**83.00**±0.73	**86.06**±0.30	10.53±0.05	**92.26**±0.63	**95.90**±0.07
Original CIFAR-10	48.49±0.17	82.70±0.38	85.47±0.33	11.43±0.59	92.01±0.44	95.75±0.08
RandomInit	9.99±0.18	37.44±1.51	50.80±0.26	7.78±1.01	84.34±0.99	93.79±0.14

Table 4: 2×2 component ablation on MNIST (MLP); active terms use λ_{cons}=1.2, λ_{drift}=0.25 with EMA auto-scaling. The bold low-accuracy entry marks the one-step collapse caused by drift without consistency; only the full configuration achieves both the best 1-step accuracy and the lowest MaxIoU.

Configuration	$\mathcal{L}_{\text{cons}}$	$\mathcal{L}_{\text{drift}}$	1-step Acc. (%)	100-step Acc. (%)	MaxIoU (↓)
Base MeanFlow	✗	✗	95.68±0.20	95.65±0.12	0.725±0.024
+ consistency only	✓	✗	95.19±2.53	95.91±0.08	0.684±0.016
+ drift only	✗	✓	**10.48**±2.88	95.88±0.51	0.711±0.080
Full DriftWeight	✓	✓	**96.30**±0.14	**96.38**±0.10	**0.661**±0.022

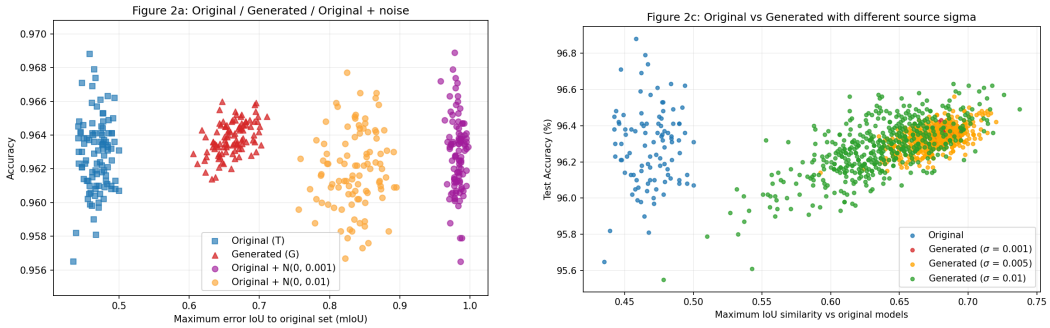
on STL-10 and 95.90% vs. 95.75% on SVHN. Random initialization remains far behind, especially on STL-10, indicating that the generated weights preserve useful transfer structure from the source population. The reported DeepWeightFlow and FLoWN rows provide prior-work context but are not treated as controlled head-to-head comparisons.

5.4 Main results: error-set diversity

Following Gupta et al. [2026], we measure the *Maximum IoU (MaxIoU)* of wrong predictions (0 = completely different errors, 1 = identical misclassifications), using leave-one-out MaxIoU over the training set as a reference (Def. 1). Figure 2a plots MaxIoU vs. accuracy on MNIST: DriftWeight concentrates at MaxIoU ≈ 0.66 (mean 0.661 ± 0.022), well separated from weight-space noise baselines (σ=0.01: ≈ 0.85) and much closer to the independent-training reference (0.47 ± 0.02). DeepWeightFlow reports 0.82 ± 0.04 on the same task [Gupta et al., 2026], overlapping with the noise baselines—functional mode collapse. A source-noise sweep (Figure 2b) further shows that DriftWeight translates input-space variation into lower error-set overlap: with σ=0.001 models cluster at MaxIoU 0.65–0.70, and as σ grows to 0.01 the population spreads down to ≈ 0.48 while keeping accuracy competitive. DeepWeightFlow, by contrast, maps diverse inputs to a narrow corridor regardless of σ (Fig. 2 in Gupta et al. 2026). MaxIoU measures error-set overlap between generated models and source checkpoints. A lower value indicates reduced overlap with the observed population, but does not by itself establish diversity among generated models, ensemble gains, or improved uncertainty estimation. Under this collapse-sensitive criterion, DriftWeight substantially reduces functional overlap on MNIST while preserving accuracy. Appendix D.4 reports PCA–function-space correlation checks.

5.5 Ablation study

We toggle the two auxiliary terms $\mathcal{L}_{\text{cons}}$ and $\mathcal{L}_{\text{drift}}$ while keeping other hyperparameters fixed, giving a 2×2 ablation on MNIST; for each setting we sample 100 models and report 1-step and 100-step accuracy together with MaxIoU (100-step samples; reference 0.47 ± 0.02).



(a) MaxIoU vs. accuracy.

(b) Source-noise σ sweep.

Figure 2: Error-set functional diversity of DriftWeight on MNIST. **(a)** MaxIoU vs. accuracy. **Blue**: original models (leave-one-out reference); **Red**: DriftWeight (1-step); **Purple/Orange**: weight-space noise baselines. DriftWeight (MaxIoU ≈ 0.66) is well separated from the noise cluster, whereas DeepWeightFlow overlaps with σ=0.01 at MaxIoU ≈ 0.82 [Gupta et al., 2026]. **(b)** Effect of source-noise scale σ: larger σ yields lower MaxIoU (less overlapping error sets) while maintaining competitive accuracy.

Main observation: a non-trivial synergy. Table 4 reveals an asymmetric, non-additive interaction between the two terms. (i) Base MeanFlow reaches 95.68% at one step but has the highest MaxIoU (0.725)—functional mode collapse. (ii) Adding $\mathcal{L}_{\text{cons}}$ alone is near-neutral in mean accuracy and mildly improves MaxIoU, but destabilizes 1-step accuracy (std 2.53). (iii) Adding $\mathcal{L}_{\text{drift}}$ alone is *catastrophic* at one step: accuracy drops to 10.48% (chance for 10 classes), yet the same model recovers to 95.88% at 100 steps. (iv) Only with both terms does DriftWeight reach 96.30% at one step *and* the lowest MaxIoU (0.661).

Boundary collapse confirms the analysis. Row (iii) is exactly the behavior predicted by the boundary-localization argument (Section 4.5). The drift loss is applied at $x_{\text{gen}} = \hat{x}_{1\text{step}}$, so its off-manifold target directly perturbs the local supervision at $u_\theta(z_1, 1, 0)$. Interior times are not directly touched by drift, explaining why 100-step integration remains accurate while one-step inference collapses; Fashion-MNIST and CIFAR-10 ResNet-20 follow-ups show the same pattern (Appendix D, Table 10). Additional ablations on temperature, PCA fitting source, negative-sampling categories, and auto-scaling are deferred to Appendix D (Table 11).

Efficiency. Under the reported H100 timing protocol, DriftWeight preserves the NFE-driven generator/solver latency advantage: at 100 steps it is $2.87\times$ faster than 100-step RK4 DeepWeightFlow, and at one step it is 243–$244\times$ faster. Per-model latency is 0.295 ms vs. 71.79 ms on MNIST and 0.293 ms vs. 71.39 ms on CIFAR-10. This comparison excludes preparation costs, inverse PCA, weight loading, and BatchNorm recalibration. Full numbers and cost accounting are in Appendix D (Table 12).

6 Conclusion

We presented DriftWeight, a one-step neural network weight generator that augments MeanFlow with a temperature-scaled repulsive drifting loss in PCA-compressed weight space, targeting *functional mode collapse*—high error-set overlap with source checkpoints despite strong generated-model accuracy. DriftWeight combines repulsive drift, checkpoint-augmented PCA, and one-step MeanFlow; our controlled ablation shows that, in this setting, drift localized at the boundary evaluation $u_\theta(z_1, 1, 0)$ must be balanced by an on-manifold term at the same point. Removing consistency collapses 1-step accuracy by 85 pp while leaving multi-step accuracy intact. In image-classification benchmarks, DriftWeight matches or exceeds reported 100-step DeepWeightFlow accuracy in a single forward pass. The direct comparative MaxIoU improvement is established on MNIST (≈ 0.66 vs. 0.82), while the ViT-Base experiment demonstrates generation at the $\sim$86M-parameter scale.

Limitations and future work. Scaling beyond the ViT-Base regime (Appendix D.7), e.g. to billion-parameter models, remains open. Checkpoint-augmented PCA incurs storage overhead: $N{=}100$ seeds $\times$ $K{=}50$ ResNet-18 checkpoints is $\approx 224\,\text{GB}$, with cost linear in N, K and model size; reducing K or using low-rank checkpoint deltas is a natural next step. Our main diversity claim relies on MaxIoU as a collapse-sensitive proxy; predictive complementarity sufficient for ensemble gains is a stronger requirement and remains future work. Our ablations do not fully isolate the effects of the PCA fitting set, retained dimensionality, and history-based negatives. PCA is a scalable proxy space rather than a functionally optimal projection; a natural extension is to learn a function-aligned projection by matching projected weight-space distances to prediction distances on a reference set, such as symmetric KL, disagreement, or error-set distance. Such projections would add reference-data dependence, pairwise prediction cost, and task specificity. Richer function-space metrics, OOD evaluation, and applications to model merging remain promising directions, as does combining repulsive drift with canonicalization [Ainsworth et al., 2023]. Our empirical scope is unconditional weight generation for image classification; other modalities and conditional generation remain future work.

Acknowledgments and Disclosure of Funding

This work received no third-party funding or support. The authors declare no additional competing interests beyond their listed affiliations.

References

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018.

Mingyang Deng, He Li, Tianhong Li, Yilun Du, and Kaiming He. Generative modeling via drifting. *arXiv preprint arXiv:2602.04770*, 2026.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations*, 2022.

Ege Erdogan. Geometric flow models over neural network weights. *arXiv preprint arXiv:2504.03710*, 2025.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. volume 31, 2018.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

Saumya Gupta, Scott Biggs, Moritz Laber, Zohair Shafi, Robin Walters, and Ayan Paul. Deep-weightflow: Re-basined flow matching for generating neural network weights. In *The Fourteenth International Conference on Learning Representations*, 2026.

David Ha, Andrew M. Dai, and Quoc V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2017.

Xiaolong Han, Zehong Wang, Bo Zhao, Binchi Zhang, Jundong Li, Damian Borth, Rose Yu, Haggai Maron, Yanfang Ye, Lu Yin, and Ferrante Neri. A survey of weight space learning: Understanding, representation, and generation. *arXiv preprint arXiv:2603.10090*, 2026.

Pavel Izmailov, Wesley J. Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Subspace inference for bayesian deep learning. In *Uncertainty in Artificial Intelligence*, volume 115 of *Proceedings of Machine Learning Research*, pages 1169–1179. PMLR, 2020.

Boris Knyazev, Michal Drozdzal, Graham W. Taylor, and Adriana Romero-Soriano. Parameter prediction for unseen deep architectures. In *Advances in Neural Information Processing Systems*, 2021.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Derek Lim, Theo Putterman, Robin Walters, Haggai Maron, and Stefanie Jegelka. The empirical impact of neural parameter symmetries, or lack thereof. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023.

Wesley J. Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in neural information processing systems*, 32, 2019.

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pages 25790–25816. PMLR, 2023.

Neale Ratzlaff and Fuxin Li. Hypergan: A generative model for diverse, performant neural networks. In *The 36th International Conference on Machine Learning*, pages 5361–5369, 2019.

Filippo Rinaldi, Giacomo Capitani, Lorenzo Bonicelli, Donato Crisostomi, Federico Bolelli, Elisa Ficarra, Emanuele Rodola, Simone Calderara, and Angelo Porrello. Update your transformer to the latest release: Re-basin of task vectors. In *International Conference on Machine Learning*, 2025.

Daniel Saragih, Deyu Cao, and Tejas Balaji. Flows and diffusions on the neural manifold. *arXiv preprint arXiv:2507.10623*, 2025a.

Daniel Saragih, Deyu Cao, Tejas Balaji, and Ashwin Santhosh. Flow to learn: Flow Matching on neural network parameters. In *Workshop on Neural Network Weights as a New Data Modality*, 2025b.

Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-Representations as generative models: Sampling unseen neural network weights. In *Advances in Neural Information Processing Systems*, 2022.

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *International Conference on Machine Learning*, pages 43947–43966. PMLR, 2024.

Aviv Shamsian, Aviv Navon, David W. Zhang, Yan Zhang, Ethan Fetaya, Gal Chechik, and Haggai Maron. Improved generalization of weight space networks via augmentations. In *International Conference on Machine Learning*, pages 44378–44393. PMLR, 2024.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency Models. In *International Conference on Machine Learning*, pages 32211–32252. PMLR, 2023.

Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. In *International Conference on Learning Representations*, 2025.

Kai Wang, Dongwen Tang, Boya Zeng, Yida Yin, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.

Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Recurrent Diffusion for large-scale parameter generation. *arXiv preprint arXiv:2501.11587*, 2025.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022.

Boya Zeng, Yida Yin, Zhiqiu Xu, and Zhuang Liu. Generative modeling of weights: Generalization or memorization? *arXiv preprint arXiv:2506.07998*, 2025.

Shaofeng Zhang, Xuanqi Chen, Ning Liao, Haoxiang Zhao, Xiaoxing Wang, Haoru Tan, Sitong Wu, Xiaosong Jia, Qi Fan, and Junchi Yan. Repulsor: Accelerating generative modeling with a contrastive memory bank. *arXiv preprint arXiv:2512.08648*, 2025.

A Background details

This appendix provides the general formulations of drifting and repulsive regularization that motivate the DriftWeight loss design in Section 4.

Drifting and repulsive regularization. Deng et al. [2026] introduced an auxiliary vector field $V(x)$ that moves samples away from collapse-prone regions while preserving the primary generation objective. A drifted target is defined as

$$\tilde{x} = x + \eta V(x), \tag{4}$$

where $\eta > 0$ is a step size. A feature-space objective encourages the current sample to follow this drift:

$$\mathcal{L}_{\text{drift}} = \|\phi(x) - \text{sg}(\phi(\tilde{x}))\|_2^2, \tag{5}$$

where $\phi(\cdot)$ is a feature map and $\text{sg}(\cdot)$ denotes the stop-gradient operator. The drift field is constructed from an attractive positive term and a repulsive negative term:

$$V_{p,q}(x) := V_p^+(x) - V_q^-(x), \tag{6}$$

where $V_p^+(x)$ points toward positive samples and $V_q^-(x)$ points toward negative samples, so that the overall field is repulsive with respect to negatives while preserving attraction toward positives.

Negative sampling as repulsive signal. Repulsive regularization is closely related to negative sampling. In contrastive learning, negative examples prevent collapse by pushing embeddings apart. Zhang et al. [2025] showed that richer negative sets strengthen this effect. The basic repulsive objective over representations $\{z_i\}_{i=1}^B$ and a negative bank $\{m_k\}_{k=1}^K$ is

$$\mathcal{L}_{\text{rep}} = \log \frac{1}{BK} \sum_{i=1}^{B} \sum_{k=1}^{K} \exp\left(-\frac{\|z_i - \text{sg}(m_k)\|_2^2}{\tau} \right), \tag{7}$$

where τ is a temperature. This objective decreases when representations are well separated from the negative bank. DriftWeight adapts this principle in weight space, constructing structured negatives from current generated samples and checkpoint history rather than a generic feature queue (Section 4.3).

MaxIoU as a functional-overlap proxy. Let w, w' be two classifiers over a test set $\mathcal{D}$ of size n, and let

$$E(w) = \{x \in \mathcal{D} : \hat{y}_w(x) \neq y(x)\}$$

denote the class-agnostic error set of w. Write

$$a_w = 1 - \frac{|E(w)|}{n}, \qquad a_{w'} = 1 - \frac{|E(w')|}{n}.$$

For pairs with $|E(w) \cup E(w')| > 0$, define the error-set IoU

$$J(w, w') = \frac{|E(w) \cap E(w')|}{|E(w) \cup E(w')|}$$

and the symmetric-difference rate

$$\Delta(w, w') = \frac{|E(w) \triangle E(w')|}{n}.$$

Equivalently, $\Delta(w, w')$ is the empirical probability that exactly one of w, w' errs on a test example. Let

$$u = \frac{|E(w) \cup E(w')|}{n}, \qquad i = \frac{|E(w) \cap E(w')|}{n}.$$

Since $J = i/u$, we have

$$\Delta(w, w') = u - i = u(1 - J).$$

Moreover,

$$(1 - a_w) + (1 - a_{w'}) = \frac{|E(w)| + |E(w')|}{n} = u + i = u(1 + J).$$

Therefore,

$$\Delta(w, w') = \frac{(2 - a_w - a_{w'})(1 - J(w, w'))}{1 + J(w, w')}. \tag{8}$$

Thus, for similarly accurate high-accuracy classifiers, high error-set IoU directly limits the fraction of examples on which exactly one classifier errs. For MaxIoU, let

$$w^\star = \arg \max_{\bar{w} \in \mathcal{W}_{\text{tr}}} J(w, \bar{w}).$$

Then Eq. (8) applies to the pair $(w, w^\star)$ with $J(w, w^\star) = \text{MaxIoU}(w)$. This is a pairwise identity; in general, one should not substitute averaged accuracies and averaged MaxIoU into Eq. (8) to obtain an exact average Δ. If $|E(w) \cup E(w')| = 0$, both classifiers are correct on all test examples and $\Delta(w, w') = 0$; this degenerate case can either be excluded or handled by the convention $J(w, w') = 1$.

This identity concerns class-agnostic error masks only. It does not imply closeness of full predictive distributions: two classifiers can share the same error set while assigning different wrong classes, probabilities, or margins. We therefore use MaxIoU as a collapse-sensitive proxy for functional overlap with observed checkpoints, not as an ensemble-utility metric or a complete functional-distance metric. Appendix D.4 reports a PCA–function diagnostic to check whether the proxy geometry carries useful functional signal.

On the correspondence between PCA distance and functional distance. Let $P \in \mathbb{R}^{k \times d}$ be the PCA projection whose rows are orthonormal principal directions, let $\Pi = P^\top P$ be the corresponding orthogonal projector, and let $x = P(w - \mu)$ and $x_0 = P(w_0 - \mu)$ denote PCA codes. For a C^2 scalar functional $F : \mathbb{R}^d \to \mathbb{R}$, Taylor expansion around a training anchor $w_0 \in \mathcal{W}_{\text{fit}}$ gives

$$F(w) - F(w_0) = \nabla F(w_0)^\top (w - w_0) + \mathcal{O}(\|w - w_0\|^2). \tag{9}$$

Decomposing the displacement into its retained PCA component and orthogonal residual yields

$$\begin{aligned} F(w) - F(w_0) = {} & (P \nabla F(w_0))^\top (x - x_0) \\ & + \nabla F(w_0)^\top (I - P^\top P)(w - w_0) + \mathcal{O}(\|w - w_0\|^2). \end{aligned} \tag{10}$$

Thus, PCA coordinates determine the first-order variation only up to the omitted-subspace term. This term is small when the compared weights lie close to the fitted PCA affine subspace, or when the gradient of F has little component in the omitted directions.

This argument does not imply that Euclidean distance in PCA space, $\|x - x_0\|_2$, is generally a first-order approximation to functional distance. For scalar F, the first-order change depends on the alignment between $x - x_0$ and $P\nabla F(w_0)$. For a vector-valued prediction map $f(w)$, write $\delta = w - w_0$. The corresponding Taylor expansion decomposes as

$$\begin{aligned} f(w) - f(w_0) = J_f(w_0)P^\top(x - x_0) \\ + J_f(w_0)(I - P^\top P)(w - w_0) + \mathcal{O}(\|w - w_0\|^2), \end{aligned} \tag{11}$$

where $x - x_0 = P\delta$. When the displacement is restricted to the retained PCA subspace, or when the omitted PCA residual is negligible, the induced local squared functional distance is approximated by the Jacobian-dependent metric

$$(x - x_0)^\top \left(PJ_f(w_0)^\top J_f(w_0)P^\top\right)(x - x_0), \tag{12}$$

rather than by Euclidean PCA distance alone.

PCA-space Euclidean distance is therefore interpreted as a practical proxy for functional variation under additional assumptions: the omitted PCA residual is small, and the local prediction Jacobian is not extremely anisotropic within the retained subspace. This is a heuristic modeling choice, not a general mathematical guarantee. For nonsmooth quantities such as prediction disagreement or error-set IoU, the Taylor argument does not apply directly; their relationship to PCA distance requires margin-style assumptions or empirical validation. Appendix D.4 provides such a diagnostic for the main ResNet-20 setting. Checkpoint-augmented PCA raises the sample-count upper bound on PCA rank from $N-1$ to $NK_{\mathrm{PCA}}-1$ (Section 4.2); the omitted residual depends on the fitted basis and the retained dimension. Learning a projection aligned to prediction distances on a reference set is a natural future direction, but would trade task-agnostic scalability for additional data and pairwise prediction computation.

B Method details

Finite-sample drift field. Section 4.3 describes the drift loss at a high level. The implementation first computes normalized distances in the PCA learning space,

$$D^+_{ij} = \|\phi(x_i) - \phi(y^+_j)\|_2, \qquad D^-_{ik} = \|\phi(x_i) - \phi(y^-_k)\|_2, \tag{13}$$

and converts them to temperature-scaled logits

$$L^+_{ij} = -\frac{D^+_{ij}}{T}, \qquad L^-_{ik} = -\frac{D^-_{ik}}{T}. \tag{14}$$

Writing $L = [L^+, L^-]$, we construct a doubly-normalized affinity matrix

$$A^{\mathrm{row}} = \mathrm{softmax}(L, \mathrm{dim} = -1), \qquad A^{\mathrm{col}} = \mathrm{softmax}(L, \mathrm{dim} = -2), \tag{15}$$

$$A = \sqrt{A^{\mathrm{row}} \odot A^{\mathrm{col}}}, \tag{16}$$

where $\odot$ is elementwise product. Row normalization makes each generated sample attend to its most relevant positives and negatives; column normalization downweights negatives that are close to many samples, preventing dense clusters from dominating the drift. Splitting $A = [A^+, A^-]$, we reweight columns by row sums and define

$$W^+_{ij} = A^+_{ij}\sum_k A^-_{ik}, \quad W^-_{ik} = A^-_{ik}\sum_j A^+_{ij}, \qquad \hat{V}_T = W^+Y^+ - W^-Y^-. \tag{17}$$

Finally, we evaluate a list of temperatures $\mathcal{T} = \{T_1, \ldots, T_M\}$ and aggregate normalized directions as $V_{\mathrm{agg}} = \sum_{T\in\mathcal{T}} \tilde{V}_T$, giving equal weight to different neighborhood scales rather than raw drift magnitudes.

Hard-negative refinement. Let D^- denote the matrix of normalized distances between the current batch and the full negative pool. For each negative candidate k, we define its hardness as

$$h_k = \min_i D^-_{ik}, \tag{18}$$

so smaller values correspond to negatives closer to at least one generated sample. DriftWeight always keeps the current generated batch at the head of the negative pool and selects the remaining negatives by the smallest hardness values, concentrating the repulsive signal on the most relevant neighbours.

Algorithm 1 DriftWeight training

Require: final checkpoints $\mathcal{W}_{\text{final}}$, history checkpoints $\mathcal{W}_{\text{history}}$, temperatures $\mathcal{T}$, coefficients $\lambda_{\text{cons}}, \lambda_{\text{drift}}$, target ratio ρ

Fit PCA on $\mathcal{W}_{\text{fit}} = \mathcal{W}_{\text{final}} \cup \mathcal{W}_{\text{history}}$
Transform final checkpoints and history checkpoints into the PCA learning space
for each training iteration **do**
    Sample a target batch x_1 from transformed final checkpoints
    Sample noise e and time pair (r, t); compute the MeanFlow base loss $\mathcal{L}_{\text{base}}$
    Generate one-step samples x_{gen} from noise
    Build positives Y^+ from the target batch
    Build negatives Y^- from current samples, same-seed history, and other-seed history
    Select hard negatives from the negative pool while keeping current samples at the head
    for each temperature $T \in \mathcal{T}$ **do**
        Compute pairwise logits, doubly-normalized affinities, and the drift field $\hat{V}_T$
    end for
    Aggregate $V_{\text{agg}} = \sum_{T \in \mathcal{T}} \tilde{V}_T$
    Compute drifting loss $\mathcal{L}_{\text{drift}}$
    Compute the one-step consistency loss $\mathcal{L}_{\text{cons}}$
    Compute $\lambda_{\text{drift,eff}}$ by `ema`, `grad`, or `none`
    Optimize $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{base}} + \lambda_{\text{cons}}\mathcal{L}_{\text{cons}} + \lambda_{\text{drift,eff}}\mathcal{L}_{\text{drift}}$
end for

Auto-scaling variants. The default EMA strategy sets the effective drift coefficient to

$$\lambda_{\text{drift,eff}} = \lambda_{\text{drift}} \cdot \frac{\text{EMA}(\mathcal{L}_{\text{base}} + \lambda_{\text{cons}}\mathcal{L}_{\text{cons}})}{\text{EMA}(\mathcal{L}_{\text{drift}}) + \epsilon} \cdot \rho, \tag{19}$$

where ρ is a target drift ratio. We also consider two alternatives: a fixed drift coefficient and a gradient-norm strategy that adjusts the coefficient by the ratio of gradient norms:

$$\lambda_{\text{drift,eff}} = \lambda_{\text{drift}} \cdot \frac{\|\nabla(\mathcal{L}_{\text{base}} + \lambda_{\text{cons}}\mathcal{L}_{\text{cons}})\|}{\|\nabla\mathcal{L}_{\text{drift}}\| + \epsilon} \cdot \rho. \tag{20}$$

We use EMA scaling as our loss-balancing implementation choice for all main experiments, based on its stability in preliminary runs.

C Experimental details and reproducibility

This appendix records the experimental conditions used for the reported DriftWeight runs, following the level of detail typically needed to reproduce weight-generation experiments. Unless otherwise stated, all datasets use their standard train/test splits, all generator runs use random seed 42, and all generated-model statistics are computed over 100 sampled weights.

Target-model populations. Table 5 summarizes how the target checkpoints were produced before training the weight generator. Final checkpoints are the supervised targets for the MeanFlow loss; intermediate checkpoints are used for checkpoint-augmented PCA and structured negative sampling in runs that use the full optimization history. For benchmark populations overlapping DeepWeightFlow, the core target-model optimizer, schedule, batch-size, and epoch settings follow the dataset-generation hyperparameters reported by Gupta et al. [2026]; unlike their released ResNet-18 script, our reported source population uses final checkpoints rather than best-checkpoint early stopping.

Generator training. All DriftWeight generators are MeanFlow models trained in the PCA learning space with AdamW, weight decay 10^{-5}, dropout 0.1, and JVPs computed with PyTorch autograd. The MeanFlow base loss samples (r, t) from either a uniform or lognormal schedule, with additional controls for how often samples lie on the flow-matching diagonal $r = t$ or at the one-step boundary $t = 1$. The MNIST and CIFAR-10 (ResNet-18) experiments use 25% diagonal samples ($r = t$) and force 37.5% of samples to the one-step boundary ($t = 1$); Fashion-MNIST and CIFAR-10 (ResNet-20) use the same diagonal ratio with 20% and 30% one-step boundary ratios, respectively. All main runs use loss exponent $\gamma = 0.25$ in the MeanFlow objective. Table 6 gives the remaining exact settings for the main reported runs.

Table 5: Training data used to construct the weight populations. The reported source accuracy is the mean±std over the independently trained seeds used as the generator training set.

Dataset	Target architecture	Target-model training recipe	Checkpoint population	Source acc.
MNIST	MLP, 784-32-32-10 ($\approx$26.5K params)	100 seeds; Adam, LR 10^{-3}; batch 64; 5 epochs	final checkpoints plus history saved every 100 iterations; 4,700 PCA-fit checkpoints in the main run	$96.31_{\pm 0.19}$
Fashion-MNIST	MLP, 784-128-128-10 ($\approx$118K params)	100 seeds; AdamW, LR 10^{-3}; batch 128; 25 epochs	final checkpoints plus history saved every 100 iterations; 11,800 PCA-fit checkpoints in the main run	$89.32_{\pm 0.17}$
CIFAR-10	ResNet-20 ($\approx$0.27M params)	100 seeds; Adam, LR 10^{-3}; batch 128; 5 epochs	final checkpoints plus history saved every 100 iterations; 2,000 PCA-fit checkpoints in the main run	$74.33_{\pm 1.83}$
CIFAR-10	ResNet-18 ($\approx$11.2M params)	DeepWeightFlow-style optimizer/schedule: 100 seeds; cosine LR; SGD, LR 0.1; weight decay 5×10^{-4}; batch 128; 100 epochs	final checkpoints plus periodic history; BatchNorm statistics are recalibrated at evaluation	$93.44_{\pm 0.22}$
CIFAR-100	ViT-Base ($\approx$85M params)	100 seeds; 100 epochs; batch 128; LR 3×10^{-4}; cosine decay with 5-epoch warmup; weight decay 0.05; dropout 0.1; label smoothing 0.1; mixup $\alpha = 0.8$; gradient clipping 1.0; AMP	20 checkpoints per seed, saved every 2,000 iterations; 2,000 PCA-fit checkpoints	$54.89_{\pm 1.22}$

Table 6: DriftWeight generator hyperparameters. "PCA fit" indicates whether the PCA basis was fitted on terminal checkpoints only or on terminal plus history checkpoints. "Neg./hard" reports the total negative-pool budget and the hard-negative subset size.

Dataset/arch	Iters	Batch	LR	Hidden	Time emb.	Source	PCA fit / dim.	Drift and consistency
MNIST/MLP	15K	16	10^{-3}	1024	64	σ=0.001, std 0.01	all / 4,699, 99.9999915%	λ_{cons}=1.2, λ_{drift}=0.25, ρ=0.1, neg./hard 2048/512, same 0.25
Fashion-MNIST/MLP	15K	16	10^{-3}	1024	64	σ=0.001, std 0.01	all / 500, 99.5708%	λ_{cons}=1.2, λ_{drift}=0.4, ρ=0.1, neg./hard 2048/256, same 0.25
CIFAR-10/ResNet-20	15K	16	10^{-3}	1024	64	σ=0.001, std 0.01	all / 1,999, $\approx$ 100.0%	λ_{cons}=1.2, λ_{drift}=0.25, ρ=0.15, neg./hard 2048/512, same 0.25
CIFAR-10/ResNet-18	15K	16	10^{-3}	4096	128	σ=0.001, std 0.01	all / 500, 100.0%	λ_{cons}=0.5, λ_{drift}=0.25, ρ=0.1, neg./hard 2048/256, same 0.25
CIFAR-100/ViT-Base	30K	16	10^{-3}	4096	128	σ=0.01, std 1.0	all / 200, 99.9999940%	λ_{cons}=0.03, λ_{drift}=0.05, ρ=0.12, neg./hard 2048/256, same 0.40

Shared drift and evaluation settings. All main DriftWeight runs use the multi-temperature drift set $\mathcal{T} = \{0.02, 0.05, 0.2\}$, EMA drift scaling with decay 0.99 and stabilizer 10^{-8}, feature/drift normalization with $\epsilon = 10^{-6}$, and a 2% history-tail exclusion except for the Fashion-MNIST, CIFAR-10 ResNet-18, and ViT-Base runs, where the full history is eligible. History negatives are split into same-seed and other-seed groups according to the same-seed ratios in Table 6; current generated samples are always included in the negative pool before hard-negative refinement.

Evaluation samples 100 weights from the trained generator with source standard deviation 0.01 for MNIST/Fashion-MNIST/CIFAR-10 and 1.0 for CIFAR-100 ViT-Base. Multi-step references use Euler integration of the MeanFlow average velocity at steps $\{1, 5, 100\}$, and apply the inverse PCA transform before loading weights into the target architecture. MNIST and Fashion-MNIST evaluation use batch size 64; CIFAR-10 evaluation uses batch size 128. For ResNet-20 and ResNet-18, BatchNorm running statistics are recalibrated on the CIFAR-10 evaluation loader, matching the DeepWeightFlow-style evaluation protocol. Inference timing is measured on an NVIDIA H100 80GB GPU with 5 warmup runs and 100 timed runs. The timing table reports generator/solver latency

only; inverse PCA, weight loading, and BatchNorm recalibration are excluded for both MNIST and CIFAR-10 where applicable.

D Additional experiments

D.1 Fashion-MNIST and ResNet-20 results

Table 7: Results on Fashion-MNIST (MLP) and CIFAR-10 (ResNet-20). We compare against DeepWeightFlow [Gupta et al., 2026] only, as no multi-method comparison tables exist for these settings.

Method	Original	Generated	Steps
Fashion-MNIST (MLP)			
DeepWeightFlow [Gupta et al., 2026]	$89.24_{\pm 0.27}$	$89.10_{\pm 0.29}$	100
DriftWeight (ours)	$89.32_{\pm 0.17}$	$\mathbf{89.87}_{\pm 0.06}$	**1**
CIFAR-10 (ResNet-20)			
DeepWeightFlow [Gupta et al., 2026]	$73.62_{\pm 2.24}$	$75.07_{\pm 1.24}$	100
DriftWeight (ours)	$74.33_{\pm 1.83}$	$\mathbf{77.97}_{\pm 0.19}$	**1**

On Fashion-MNIST, DriftWeight at one step (89.87%) exceeds DeepWeightFlow at 100 steps (89.10%). On CIFAR-10 ResNet-20, DriftWeight (77.97%) surpasses both DeepWeightFlow (75.07%) and its own training mean (74.33%). This setting has the smallest ratio of training models to parameter dimension, making it especially susceptible to memorization.

D.2 Multi-step reference

Table 8: DriftWeight accuracy with additional generation steps. Although designed for one-step generation, the MeanFlow backbone also supports multi-step integration.

Steps	MNIST (MLP)	F-MNIST (MLP)	CIFAR-10 (ResNet-18)	CIFAR-10 (ResNet-20)
1 (default)	$96.30_{\pm 0.14}$	$89.87_{\pm 0.06}$	$93.64_{\pm 0.12}$	$77.97_{\pm 0.19}$
5	$96.35_{\pm 0.10}$	$89.85_{\pm 0.04}$	$93.64_{\pm 0.06}$	$78.08_{\pm 0.11}$
100	$96.38_{\pm 0.10}$	$89.82_{\pm 0.03}$	$93.75_{\pm 0.02}$	$78.13_{\pm 0.10}$

Additional integration changes accuracy only modestly; the default one-step results remain close to the corresponding 100-step references.

D.3 Transfer fine-tuning protocol

The transfer results in Table 3 use the released DeepWeightFlow transfer protocol with three initializer sources: one-step DriftWeight samples, independently trained CIFAR-10 ResNet-18 source checkpoints, and random initialization. For each target dataset, five initializers are evaluated at epoch 0 and after 1 or 5 fine-tuning epochs. Generated ResNet-18 weights first receive CIFAR-10 BatchNorm recalibration; target inputs are resized to 32×32 and normalized with the STL-10 or SVHN statistics; STL-10 labels keep the native torchvision class order; and fine-tuning uses AdamW (learning rate 10^{-4}, weight decay 10^{-5}) with cosine decay. For generated and original CIFAR-10 initializers, the first 40% of the ResNet feature blocks are frozen during fine-tuning; random initializers are fully trainable. Reported epoch-1 and epoch-5 numbers are best-so-far target test accuracies within the corresponding fine-tuning budget, matching the DeepWeightFlow implementation.

The generated initializers are sampled from the same CIFAR-10 ResNet-18 DriftWeight model used for the complete-generation results in Table 2. This keeps the transfer experiment tied to the main complete-weight generation setting rather than using a separately tuned transfer-specific generator.

D.4 PCA–function diagnostics

This diagnostic checks whether the geometry used by the drift loss is aligned with observable functional differences. For CIFAR-10 ResNet-20, pairs are formed between generated models and

source checkpoints; each pair has one Euclidean distance in PCA space and several prediction-based distances on the test set. Pearson correlation measures whether larger PCA distances tend to correspond to larger functional distances in scale, while Spearman correlation checks whether the pairwise rankings are similar. The diagnostic uses the saved 100-step ResNet-20 samples and is therefore treated as auxiliary evidence rather than part of the one-step main claim.

Table 9: Correlation between PCA-space distance and function-space distances on CIFAR-10 ResNet-20, using $5{,}000$ generated–source model pairs. Each row asks whether the same PCA distance is associated with a particular prediction-based distance on the test set.

Function-space distance	Pearson r	Spearman ρ
Prediction disagreement	0.27	0.18
Symmetric KL	0.28	0.19
$1 - \mathrm{IoU}$ of error sets	0.50	0.13

PCA distance therefore carries useful signal about functional differences in the ResNet-20 setting, especially error-set distance, with positive rank correlations across all three metrics. The correlations are still moderate rather than definitive, so we treat PCA repulsion as a practical geometric proxy whose validity should be checked empirically.

D.5 Additional ablations

Fashion-MNIST and CIFAR-10 ResNet-20 component ablations. The same 2×2 component check on Fashion-MNIST and CIFAR-10 ResNet-20 reproduces the boundary-localized failure beyond MNIST. Drift alone sends 1-step accuracy to chance level, while longer integration recovers, and the full objective gives the best one-step result in both settings.

Table 10: 2×2 component ablations on Fashion-MNIST (MLP) and CIFAR-10 ResNet-20. Removing the consistency term while keeping drift collapses 1-step accuracy to chance level in both settings, even though 100-step integration recovers to the normal accuracy range.

			Fashion-MNIST (MLP)		CIFAR-10 (ResNet-20)	
Configuration	$\mathcal{L}_{\text{cons}}$	$\mathcal{L}_{\text{drift}}$	1-step Acc. (%)	100-step Acc. (%)	1-step Acc. (%)	100-step Acc. (%)
Base MeanFlow	✗	✗	$89.56_{\pm 0.05}$	$89.64_{\pm 0.15}$	$77.17_{\pm 0.22}$	$77.41_{\pm 0.42}$
+ consistency only	✓	✗	$89.62_{\pm 0.06}$	$89.74_{\pm 0.04}$	$77.30_{\pm 0.13}$	$77.42_{\pm 0.09}$
+ drift only	✗	✓	$10.00_{\pm 0.00}$	$89.63_{\pm 2.30}$	$9.93_{\pm 0.87}$	$77.09_{\pm 0.69}$
Full DriftWeight	✓	✓	$\mathbf{89.87}_{\pm 0.06}$	$\mathbf{89.82}_{\pm 0.03}$	$\mathbf{77.97}_{\pm 0.19}$	$\mathbf{78.13}_{\pm 0.10}$

Note on the std=0 entry. We verified by re-generating from the checkpoint that the Fashion-MNIST "drift only" 1-step value is not a numerical artifact. Fashion-MNIST has a balanced test set with exactly 1,000 examples per class, so predicting any single class for all 10,000 test images gives exactly 10.00% accuracy. The 100 generated weights are distinct (pairwise ℓ_2 distances $\approx$6–8), contain no NaN/Inf values, and have normal statistics (std $\approx$0.026, $\|w\|_2 \approx$9–10); logits are finite and in a normal range (about $[-1.7, -0.4]$). All 100 models nevertheless predict class 6 ("Shirt") for every test example, yielding $10.00_{\pm 0.00}\%$. This is the boundary-localized failure predicted by Section 4.5: the drift objective (Eq. 1) acts only at $x_{\text{gen}} = \hat{x}_{1\text{step}}$, and without the consistency anchor (Eq. 2) only off-manifold pressure remains at $u_\theta(z_1, 1, 0)$. The same model recovers to 89.63% at 100 steps, showing that the interior flow remains usable; compared with MNIST ($10.48_{\pm 2.88}\%$), Fashion-MNIST collapses more uniformly to a single shirt-like constant classifier, strengthening the boundary-localization interpretation.

These two follow-ups are consistent with the same mechanism: repulsive drift improves accuracy only when paired with the one-step consistency anchor; without that anchor, the generator fails at the 1-step evaluation point and recovers only under additional integration.

Temperature design. We isolate the role of the multi-temperature drift set $\mathcal{T} = \{0.02, 0.05, 0.2\}$ by retraining with each temperature in isolation, keeping every other hyperparameter at the reference

DriftWeight configuration (Sec. 5.1). Table 11 reports both 1-step and 100-step accuracy together with MaxIoU.

Table 11: Effect of temperature configuration on MNIST (DriftWeight full objective; single-temperature runs use only the indicated T). Multi-T is the reference DriftWeight setting $\mathcal{T} = \{0.02, 0.05, 0.2\}$. MaxIoU is measured over 100 generated models; the leave-one-out training reference is 0.47 ± 0.02. Accuracy statistics below ∼90% (greyed) render MaxIoU difficult to interpret, because a high error rate by itself decorrelates wrong-prediction masks and inflates apparent error-set diversity.

Temperature	1-step Acc. (%)	100-step Acc. (%)	MaxIoU (↓)
$T = 0.02$ only	$96.10_{\pm 0.14}$	$96.27_{\pm 0.06}$	$0.820_{\pm 0.024}$
$T = 0.05$ only	$95.25_{\pm 0.98}$	$95.39_{\pm 0.14}$	$0.749_{\pm 0.022}$
$T = 0.2$ only	$58.00_{\pm 21.65}$	$93.26_{\pm 1.92}$	$0.445_{\pm 0.182}$
Multi-T (default)	$\mathbf{96.30}_{\pm 0.14}$	$\mathbf{96.38}_{\pm 0.10}$	$\mathbf{0.661}_{\pm 0.022}$

Each single temperature exhibits a distinct failure mode. A small T=0.02 yields strong *local* repulsion only—accuracy is competitive (96.10%), but the generated population cannot escape neighborhood-level clusters and MaxIoU rises to 0.820, approximately the DeepWeightFlow regime. A large T=0.2 gives diffuse global repulsion strong enough to push generated weights off the valid-weight manifold: 1-step accuracy collapses to $58.0 \pm 21.7\%$ while 100-step accuracy also degrades to 93.3%. The apparent MaxIoU improvement (0.445) is largely an artifact of the high error rate—at chance-level accuracy, wrong-prediction masks become nearly uncorrelated regardless of the underlying functional similarity—so this operating point does not meet the joint accuracy–MaxIoU criterion. Combining all three temperatures recovers the best of both regimes: MaxIoU improves from 0.820 (T=0.02 only) to 0.661 while 1-step accuracy is *also* the highest at 96.30%, demonstrating that multi-scale drift is strictly better than any single-scale alternative on the accuracy–MaxIoU frontier.

PCA fitting source. Fitting PCA on final checkpoints only (99 components from 100 seeds) limits the basis to $N-1$ dimensions—the same bottleneck that constrains DeepWeightFlow. Including intermediate checkpoints allows the PCA basis to extend beyond the rank limit imposed by terminal checkpoints alone.

Auto-scaling strategy. EMA scaling is the default loss-balancing choice in our implementation. Appendix B specifies the rule and the fixed-coefficient and gradient-norm alternatives. The reported configurations and ablations do not constitute an exhaustive hyperparameter search.

D.6 Generation efficiency

Integration method. To match the original DeepWeightFlow protocol [Gupta et al., 2026], our DeepWeightFlow re-implementation uses an RK4 integrator that evaluates the velocity network four times per integration step ($k_1, \ldots, k_4$), so a nominal "100-step" run corresponds to ≈ 400 network evaluations per sample. DriftWeight instead predicts the *average* velocity $u(z_t, r, t)$ and advances the state with a single forward pass per step, giving NFE equal to the step count. The efficiency gap thus stems from two compounding factors: (i) per-step solver cost (RK4 vs. single Euler update), and (ii) total step count.

DriftWeight one-step generation produces an MLP flow sample in 0.295 ms and a CIFAR-10 flow sample in 0.293 ms—243× and 244× faster than 100-step RK4 DeepWeightFlow, respectively. At equal nominal step count (100), DriftWeight is already 2.87× faster by avoiding the 4× NFE multiplier of RK4; the remaining speedup at ≤ 5 steps comes from the drastically reduced step count that MeanFlow-style average velocity enables. These measurements isolate generator/solver latency and exclude inverse PCA, weight loading, and BatchNorm recalibration. Source-model training, checkpoint storage, PCA fitting, and generator training are preparation costs amortized over repeated sampling. Checkpoint augmentation retains intermediate states from the same source training runs; it adds storage without requiring additional source-model training runs. Both DriftWeight and DeepWeightFlow require inverse PCA and model materialization, with BatchNorm recalibration

Table 12: Generator/solver latency comparison. Wall-clock time per sample measured on a single NVIDIA H100 80GB GPU, averaged over 100 runs after 5 warmup runs, excluding preparation costs, inverse PCA, weight loading, and BatchNorm recalibration. †DeepWeightFlow is our faithful re-implementation of Gupta et al. [2026]; following the original protocol it integrates the learned velocity field with a fourth-order Runge–Kutta (RK4) solver, which evaluates the network $4\times$ per integration step. DriftWeight uses the learned average velocity and performs a single network evaluation per step. "NFE" denotes the resulting number of function (network) evaluations per generated model. Speedup is relative to DeepWeightFlow 100-step on the same row. For CIFAR-10, the timing measurement should be read as a flow-generation latency comparison for the corresponding generator run, rather than as an end-to-end architecture-specific model materialization time.

Run	Method	Solver	Steps	NFE	Time/model (ms)	Speedup
MNIST (3-Layer MLP)	DeepWeightFlow†	RK4	100	400	$71.79_{\pm 1.12}$	$1\times$
	DriftWeight	Euler	100	100	$24.98_{\pm 0.20}$	$2.87\times$
	DriftWeight	Euler	5	5	$1.30_{\pm 0.01}$	$55\times$
	DriftWeight	Euler	1	1	$0.295_{\pm 0.008}$	$\mathbf{243}\times$
CIFAR-10 (ResNet-20)	DeepWeightFlow†	RK4	100	400	$71.39_{\pm 0.33}$	$1\times$
	DriftWeight	Euler	100	100	$24.91_{\pm 0.30}$	$2.87\times$
	DriftWeight	Euler	5	5	$1.30_{\pm 0.02}$	$55\times$
	DriftWeight	Euler	1	1	$0.293_{\pm 0.008}$	$\mathbf{244}\times$

where applicable. The reported speedups do not measure these shared stages. DriftWeight also incurs pairwise drift computation during generator training.

D.7 Scaling to ViT-Base on CIFAR-100

We evaluate DriftWeight on a target architecture that is roughly *three orders of magnitude larger* than the MNIST MLP used in the main experiments: a ViT-Base backbone ($\sim$86M parameters) trained on CIFAR-100 with $N{=}100$ independently trained seeds and $K_{\mathrm{PCA}}{=}20$ PCA-fit checkpoints per seed (2,000 checkpoints in total). Under this setting the checkpoint-augmented PCA ceiling rises from $N{-}1{=}99$ to $NK_{\mathrm{PCA}}{-}1{=}1{,}999$ components, allowing this run to retain 200 PCA components beyond the terminal-only rank limit.

Table 13: Scaling DriftWeight to CIFAR-100 with a ViT-Base target ($\sim$86M parameters; three orders of magnitude larger than the MLP setting). The training population comprises 100 independently trained seeds with 20 PCA-fit checkpoints each, yielding a PCA ceiling of $NK_{\mathrm{PCA}}{-}1{=}1{,}999$ components; this run retains 200 components capturing $\geq 99.9999\%$ variance. "Original" reports the mean$\pm$std of the 100 training seeds. Generated accuracies are averaged over 100 sampled models; we also report multi-step integration of the same trained DriftWeight generator at 5 and 100 Euler steps.

Method	Original	Generated	Steps
CIFAR-100 (ViT-Base, $\sim$86M params)			
DriftWeight (ours)	$54.88_{\pm 1.22}$	$\mathbf{53.21}_{\pm 0.25}$	**1**
		$53.70_{\pm 0.12}$	5
		$53.70_{\pm 0.13}$	100

Discussion. This experiment demonstrates the feasibility of one-step weight generation for a ViT-Base target. The retained 200-component basis exceeds the 99-component limit of PCA fitted on 100 terminal checkpoints. This establishes the feasibility of the larger-subspace configuration, while the isolated contribution of history augmentation and the minimum sufficient dimension remain to be characterized at this scale. Empirically, DriftWeight generates 100 ViT-Base classifiers in a single forward pass with $53.21 \pm 0.25\%$ test accuracy on CIFAR-100 (Table 13), trailing the training mean by only ≈ 1.7 pp. The same trained generator integrated with 100 Euler steps reaches $53.70 \pm 0.13\%$, closing another ≈ 0.5 pp of the gap —consistent with the smaller-scale trend (Table 8) that the MeanFlow backbone benefits from multi-step refinement on harder regimes while still delivering strong one-step performance. The present experiments do not isolate the causes of the residual accuracy gap. Varying the retained PCA dimension and generator capacity would help distinguish

compression effects from generator approximation. Scaling beyond the ViT-Base regime remains open.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (Section 1) state that DriftWeight improves one-step weight generation by preserving accuracy while reducing collapse-sensitive error-set overlap. Sections 5.2–5.4 provide empirical evidence supporting these claims.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 includes an explicit Limitations paragraph discussing scaling beyond the ViT-Base regime, storage overhead of checkpoints, and the scope of the MaxIoU-based diversity metric.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper is primarily empirical. The method is described algorithmically (Algorithm 1) and does not include formal theorems or proofs.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 5.1 details all datasets, architectures, baselines, hyperparameters, and training procedures. Algorithm 1 provides the full training loop. Code is publicly available at the repository linked in Section 1.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and instructions for source-model training, generator training, and evaluation are publicly available at the repository linked in Section 1. The release supports MNIST, Fashion-MNIST, and CIFAR-10; the CIFAR-100/ViT experiment code is not included. All datasets used (MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100) are publicly available.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 specifies all training details including optimizer, learning rate, batch size, PCA configuration, drift hyperparameters, and evaluation protocol.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Accuracy results in the main tables and appendix tables report mean $\pm$ standard deviation over 100 generated models or training seeds. The captions state the population being averaged.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5.1 describes the compute setup. All experiments were conducted on a single GPU. Training completes within the same order of magnitude as the MeanFlow baseline (Appendix C).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics. The work generates neural network weights for standard classification tasks and does not involve human subjects, private data, or dual-use concerns.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper does not include a dedicated discussion of broader societal impacts. The work is foundational research on generative modeling of neural network weights for standard benchmark classifiers, and we do not foresee direct negative societal impacts beyond those associated with standard image-classification benchmarks.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The models studied are standard benchmark classifiers for MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100, not high-risk generative models or datasets requiring special responsible-release safeguards.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets used (MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100) are publicly available standard benchmarks. Baseline numbers are properly cited from published papers.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The public repository linked in Section 1 contains documented source code, dependency versions, and training and evaluation commands.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing or human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This work does not involve human subjects.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were not used as a component of the core methodology. They were used only for writing assistance, which does not require declaration per NeurIPS policy.