HyperFlow: Gradient-Free Test-Time Adaptation for Cross-Domain Few-Shot Classification

Donggyun Kim **Chanwoo Kim** **Seunghoon Hong**
KAIST School of Computing
donggyun.kim@kaist.ac.kr, kcwkcw0802@kaist.ac.kr, seunghoon.hong@kaist.ac.kr

Abstract

While test-time fine-tuning is beneficial in cross-domain few-shot classification, the need for multiple backpropagation steps can be prohibitively expensive in resource-constrained environments. We propose HyperFlow, a gradient-free test-time adaptation method that amortizes fine-tuning dynamics into a lightweight conditional drift network. During offline training, which uses standard backpropagation, HyperFlow learns from gradient-based fine-tuning trajectories collected on meta-training tasks. Once trained, it adapts a selected PEFT parameter subspace (*e.g.,* bias or LoRA parameters) for a new task by numerical ODE solving, requiring only forward passes of the drift network and no gradient computation at test time. In experiments on Meta-Dataset and CD-FSL benchmarks, our method improves out-of-domain performance over the direct transfer approach while using only 6–14% of peak memory and about 1% of the FLOPs of standard fine-tuning, positioning HyperFlow as an accuracy–efficiency trade-off between direct transfer and fine-tuning. Code is available at `https://github.com/GitGyun/HyperFlow`.

1 Introduction

Few-shot classification aims to recognize novel classes of images using only a handful of labeled examples per class [1, 2, 3, 4]. Unlike conventional supervised learning approaches that use an extensive amount of labeled images for pre-defined tasks to train specialized models, few-shot learning approaches focus on adapting or transferring a pretrained model to novel tasks using a few examples, where the target classes and image domains can vary at test time. This paradigm is relevant for real-world scenarios where the image domain is distinctive and the labeled data is scarce or expensive to obtain, such as medical imaging [5], plant disease classification [6], and remote sensing [7]. Even in the era of increasing general-purpose foundation models [8], few-shot adaptation remains useful for improving performance in domain-specific contexts [9, 10, 11].

Recent cross-domain few-shot classification methods often benefit from an additional fine-tuning stage on the support examples provided at test time. Embedding-based methods [3, 12, 13] classify query images by learning a shared representation space and assigning labels according to distances or similarities to reference points, such as class prototypes. Although these methods can be applied by direct transfer without updating the model, prior work has shown that test-time fine-tuning can improve performance, especially when there is a domain discrepancy between training and testing tasks [14, 15, 16, 17, 18]. This creates an accuracy–efficiency trade-off: direct transfer is cheap but can be insufficient under domain shift, whereas fine-tuning is more adaptive but requires deployment-time backpropagation through the target model.

The cost of fine-tuning is not primarily due to the number of support examples, but to the repeated gradient computation and activation storage required by backpropagation through the target network. This can be undesirable in resource-constrained deployment scenarios, especially when adaptation must be performed repeatedly across many test tasks or on memory-limited devices. A natural goal is therefore to recover part of the benefit of fine-tuning while avoiding backpropagation during test-time

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

adaptation. This goal is related to meta-learning methods [19, 20, 21, 22] that replace the inner-loop update with learned gradient-free mechanisms such as synthetic gradients [23], Neural ODEs [24], or diffusion models [25]. However, these methods typically focus on adapting prototypes or linear classifiers, and often still require target-model backpropagation during training. Another related line of work studies parameter generation [26, 27, 28, 29], but most existing methods are not designed specifically for cross-domain few-shot adaptation under deployment-time efficiency constraints.

We propose *HyperFlow*, a gradient-free test-time adaptation framework for few-shot learning that emulates fine-tuning dynamics in a selected parameter-efficient subspace. HyperFlow trains a lightweight conditional drift network to predict task-dependent parameter updates from the few-shot support set. Motivated by the connection between gradient descent and gradient flows, we view fine-tuning trajectories as discretized ODE paths and train the drift network from offline trajectories collected on meta-training tasks. At deployment time, once the drift network has been trained, adaptation reduces to numerical ODE solving using only forward passes of the drift network, without backpropagation through the target model. Note that the offline trajectory collection and drift-network training use standard backpropagation. Throughout the paper, "gradient-free" refers exclusively to the adaptation on unseen test tasks. To keep the parameter dynamics tractable, we instantiate HyperFlow on compact PEFT subspaces, such as selected bias or LoRA parameters, rather than attempting to model the full parameter space of modern backbones. This design shifts the test-time adaptation cost from target-model backpropagation to lightweight drift-network inference, substantially reducing deployment-time memory and compute compared to gradient-based fine-tuning. HyperFlow therefore targets a resource-asymmetric setting, in which the one-time offline stage is performed on a resource-sufficient server, while adaptation is performed repeatedly under deployment-time memory and compute constraints.

We demonstrate HyperFlow on cross-domain few-shot classification benchmarks, including Meta-Dataset [30] and CD-FSL [14]. When applied to a strong few-shot classification pipeline [18], HyperFlow improves out-of-domain performance over direct transfer while requiring substantially less peak memory and FLOPs than standard fine-tuning. We further examine its behavior across different backbones, parameter subspaces including LoRA, and an additional few-shot adaptation framework. Overall, the results position HyperFlow as a practical intermediate point between direct transfer and gradient-based fine-tuning, providing a lightweight test-time adaptation alternative when deployment-time backpropagation is constrained.

2 Preliminary

Problem Setup. Few-shot classification (FSC) aims to recognize novel classes of images using only a few labeled examples per class. To achieve this, a model is first trained on a base dataset $\mathcal{D}_{\text{base}} = \{(x_i, y_i)\}_{i=1}^{N_{\text{base}}}$, often called the "meta-training" dataset, which consists of many labeled examples for known base classes $\mathcal{C}_{\text{base}}$. During testing, the model is evaluated on multiple classification tasks involving novel classes $\mathcal{C}_{\text{novel}}$, where $\mathcal{C}_{\text{base}} \cap \mathcal{C}_{\text{novel}} = \emptyset$. To learn the novel classes of each test task $\mathcal{T}$, the model receives a small set of labeled examples $\mathcal{S}_{\mathcal{T}} = \{(x_j, y_j)\}_{j=1}^{N_{\mathcal{T}}}$ called the *support set*. Then, the model is tested to classify unlabeled images within the same task, called the *query set*. In a conventional N-way K-shot setting [3, 31], each task consists of N classes, and K labeled images per class are given as the support set.

In this paper, we consider a more challenging *cross-domain* few-shot classification problem, where the testing images originate from domains different from those of the base dataset $\mathcal{D}_{\text{base}}$ (*e.g.,* domain shifts from natural images to X-ray). In this setup, the model must have strong generalization ability and a flexible adaptation mechanism to address both novel classes and out-of-domain images.

P>M>F Pipeline for FSC. While various approaches have been proposed to tackle the FSC problem, a common training pipeline involves one or more of three distinct stages—Pre-training, Meta-training, and Fine-tuning—collectively known as P>M>F [18]. In the pre-training stage, general-purpose representations are learned using backbones trained with self-supervised methods on large-scale datasets (*e.g.,* the DINO backbone [32] pre-trained on ImageNet [33]). During the meta-training stage, the model acquires prior knowledge about the underlying task distribution, typically by simulating few-shot learning episodes using the base dataset $\mathcal{D}_{\text{base}}$ [3]. Finally, the fine-tuning stage further adapts the model to test tasks using the provided support set, enabling flexible adaptation to novel classes and unseen domains. During the fine-tuning stage, parameter-efficient fine-tuning (PEFT) techniques [34, 35] can be applied to reduce over-fitting to the support set.

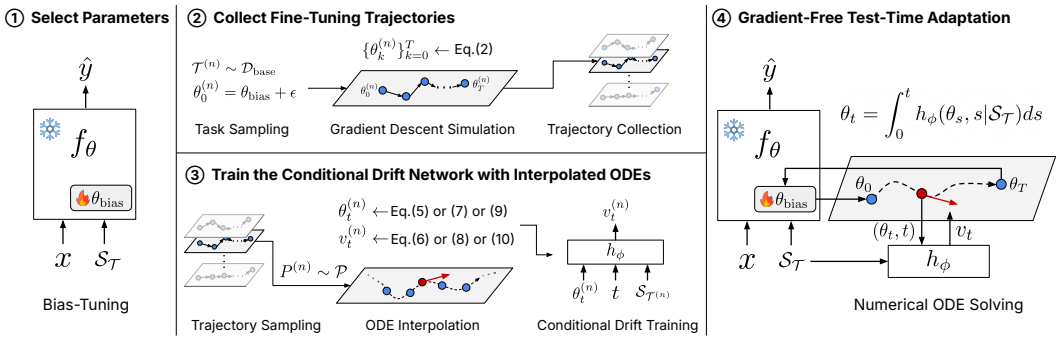


Figure 1: Overview of HyperFlow. (1) We select a compact set of target parameters, such as bias parameters of f_θ. (2) We collect offline fine-tuning trajectories on meta-training tasks. (3) We continuously interpolate the trajectories to train the conditional drift network h_ϕ. (4) At test time, HyperFlow adapts the selected parameters by numerical ODE solving with forward passes of h_ϕ, without backpropagation through the target model.

3 Method

We focus on the *fine-tuning* stage of the training pipeline of few-shot classification (FSC). While it has been consistently reported that fine-tuning is beneficial for FSC [14, 15, 16, 17, 18], especially in cross-domain scenarios, it necessitates the computation of expensive backpropagation throughout the network. This can be problematic in practical applications of FSC where only limited computational resources are available for adaptation. For efficient few-shot adaptation, we propose HyperFlow, which replaces the fine-tuning stage with a much cheaper process. See Figure 1 for an overview.

3.1 HyperFlow: Emulating Gradient Flows

Fine-tuning involves gradient descent on the model parameters. Let f_θ be the model with parameters θ and $\mathcal{L}$ be a loss function. Given a test task $\mathcal{T}$ with a corresponding support set $\mathcal{S}_\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{N_\mathcal{T}}$, the objective of few-shot adaptation is to minimize the loss over the support:

$$L_\mathcal{T}(\theta) = \sum_{(x_i,y_i)\in\mathcal{S}_\mathcal{T}} \mathcal{L}\left(f_\theta(x_i), y_i\right). \tag{1}$$

This is achieved by iterative gradient descent updates on θ from its initialization $\theta_0 = \theta_{\text{init}}$ as follows:

$$\theta_k \leftarrow \theta_{k-1} - \lambda \cdot g\left(\nabla_\theta L_\mathcal{T}(\theta_{k-1})\right), \quad k = 1, 2, \cdots, T, \tag{2}$$

where θ_k denotes the parameters at k-th iteration, λ denotes the learning rate, and g is an optimizer (*e.g.,* Adam [36]). However, in resource-constrained environments, computing the gradient $\nabla_\theta L_\mathcal{T}(\theta)$ can be burdensome, especially when the network f_θ is deep and large.

Our key idea is to emulate the gradient descent updates (Eq. 2) with an ODE-solving process that requires no test-time gradient descent. As already remarked by prior studies [37, 38, 39], the gradient descent updates can be seen as an Euler discretization of the corresponding ordinary differential equation (ODE), which is also known as the gradient flow [40]. For example, the gradient flow of task $\mathcal{T}$ with the simplest gradient descent optimizer (*i.e.,* $g(\theta) = \theta$) can be expressed as follows:

$$\frac{d\theta(t)}{dt} = -\frac{\partial L_\mathcal{T}(\theta(t))}{\partial \theta}. \tag{3}$$

Inspired by Neural ODEs [24], we approximate the ODE function using a drift network h_ϕ. Then, we can emulate the gradient descent by a numerical integration method (*e.g.,* the Euler method), which only requires several forward steps of h_ϕ.

Since the gradient flow differs by tasks, we employ a *conditional* drift network $h_\phi(\theta, t|\mathcal{S}_\mathcal{T})$ to predict the task-specific drift $\frac{d\theta(t)}{dt}$ of task $\mathcal{T}$ conditioned on the support set $\mathcal{S}_\mathcal{T}$. To train the conditional drift network, we perform an extra training stage using the base dataset as follows:

$$\min_\phi \mathbb{E}_{t,\mathcal{T}}\left[\left\| h_\phi(\theta_t, t|\mathcal{S}_\mathcal{T}) - \left(-\frac{\partial L_\mathcal{T}(\theta(t))}{\partial \theta}\right)\right\|^2\right], \tag{4}$$

where the timestep t is uniformly sampled from $[0, T]$ and the training task $\mathcal{T}$ is sampled from the base dataset $\mathcal{D}_{\text{base}}$. The drift network is trained on tasks sampled from the base dataset and is expected to transfer when the offline trajectories capture adaptation patterns shared across tasks. This assumes that tasks fine-tuned from the same initialization in the same PEFT space share a low-dimensional adaptation geometry, as suggested by prior studies on fine-tuned parameter displacements [41, 42, 43, 44]. In Appendix E, we empirically verify this shared structure and analyze how trajectory diversity affects this behavior.

3.2 Target Parameters Selection

A straightforward challenge in learning the gradient flow with a drift network is that the number of target network parameters is often too large. For example, vision transformers [45], a widely used architecture in computer vision, typically have millions to billions of parameters, making it nearly infeasible to directly treat them as inputs and outputs of another neural network. However, recent studies on parameter-efficient fine-tuning (PEFT) show that updating a tiny set of parameters is sufficient to modulate large pre-trained backbones [46, 34, 35]. Therefore, we reduce the number of parameters considered in the gradient flows by selecting a tiny subset of the whole parameters θ.

Specifically, we adopt bias-tuning [34] that updates only the bias parameters of the model f_θ as the default parameterization. In the few-shot learning context, bias-tuning has been shown to be extremely parameter-efficient and robust to over-fitting for adapting large transformers to unseen vision tasks [47, 48, 49]. Also, it adds no extra computation overhead to the original model during inference. By further selecting a subset of bias parameters (*e.g.,* those from QKV-projection layers of the attention blocks), we can reduce the number of target parameters to be sufficiently tangible for the drift network (*e.g.,* thousands), without losing much adaptability at the few-shot adaptation phase as we show in Section 5.2. However, HyperFlow is not structurally tied to bias parameters: once a PEFT parameterization is vectorized, the same training objective and ODE-based adaptation apply, and we also instantiate HyperFlow on rank-4 LoRA [35] and the adapters of TSA [50] in Section 5.4.

Once a feasible target parameter subspace is selected, HyperFlow's deployment-time adaptation no longer requires backpropagation through the target model. The adaptation cost is instead governed by the dimensionality of the selected parameter vector and the size of the drift network h_ϕ. This differs from conventional PEFT fine-tuning, where even a small adapted subspace still requires activation storage and backpropagation through the target model. Thus, HyperFlow helps translate parameter efficiency into deployment-time memory and compute efficiency. We provide a detailed computation-cost analysis in Section 5.3.

3.3 Drift Training with Continuous ODEs

Training the conditional drift network using the continuous-time ODE objective (Eq. 4) requires expensive computation of the gradient $\nabla_\theta L_{\mathcal{T}}(\theta)$ at arbitrary timestep $t \in [0, T]$ as well as the simulation of the gradient flow until t. To make the training process more efficient, we first collect a fixed number of fine-tuning trajectories by simulating the gradient descent on the base dataset. Specifically, for each task $\mathcal{T}^{(n)}$ constructed from the base dataset $\mathcal{D}_{\text{base}}$, a discretized trajectory $P^{(n)} = \{\theta_0^{(n)}, \cdots, \theta_T^{(n)}\}$ is generated by applying T gradient descent steps (Eq. 2) on the initial parameters $\theta_{\text{init}}^{(n)}$. We repeat this process multiple times by randomly perturbing the initialization $\theta_{\text{init}}^{(n)}$, yielding a dataset of trajectories $\mathcal{P} = \{P^{(n)}\}_{n=1}^{N_{\text{traj}}}$. During training, we approximate the continuous gradient flows by smoothly interpolating the discretized points to compute the intermediate parameter θ_t and its drift v_t at an arbitrary timestep t. We propose both linear and nonlinear objective ODEs that interpolate the discretized gradient flows. Below, we omit the superscript of $\theta^{(n)}$ for simplicity.

Linear Flow (HyperFlow-L). A linear flow (**HyperFlow-L**) is generated by linearly interpolating the initial and final points of each trajectory. Specifically, given the two endpoints θ_0 and θ_T of a trajectory, the point θ_t and its drift v_t (time-derivative) at timestep $t \in [0, T]$ is computed as follows:

$$\theta_t = (1 - t/T) \cdot \theta_0 + t/T \cdot \theta_T, \tag{5}$$

$$v_t = (\theta_T - \theta_0)/T. \tag{6}$$

Note that this objective does not use any intermediate points $\{\theta_k\}_{k=1}^{T-1}$. As it promotes the straightness of the learned flow, it encourages the drift network to solve the ODE with only a few Euler steps. However, the straight path emulated by Eq. (5)-(6) may deviate from the true trajectory that follows the loss surface, so the flexibility of HyperFlow can be limited by the oversimplified ODEs.

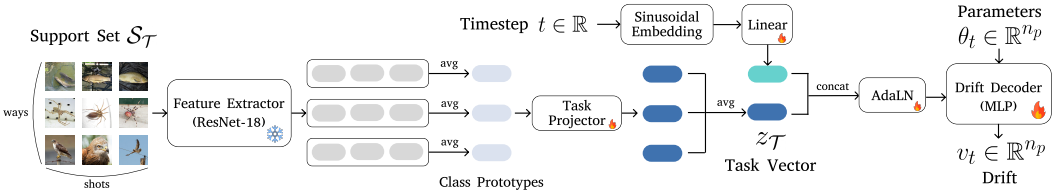


Figure 2: The conditional drift network architecture of HyperFlow.

Nonlinear Flows (HyperFlow-PL & HyperFlow-C). To cope with more complex loss surfaces, we also propose nonlinear flow objectives that exploit the full trajectories. A simple way to extend the linear flow with intermediate points is to interpolate each discrete transition $\theta_k \to \theta_{k+1}$ individually, *i.e.,* defining piecewise-linear flows (**HyperFlow-PL**). Specifically, we divide the time interval $[0, T]$ into T segments $[0, 1], \cdots, [T-1, T]$, then linearly interpolate θ_t and v_t in each segment $[k-1, k]$:

$$\theta_t = (k - t) \cdot \theta_{k-1} + (t - k + 1) \cdot \theta_k, \tag{7}$$

$$v_t = \theta_k - \theta_{k-1}. \tag{8}$$

However, such non-smooth transitions in the velocity field are difficult to model by the conditional drift network and can cause over-fitting to training trajectories. Thus, we create smooth ODEs that pass through the discrete points using cubic Hermite splines [51, 52]. To compute the interpolated point and its drift, the cubic flow (**HyperFlow-C**) defines local cubic curves passing four neighboring points at each timestep. Let $\tau = t - k + 1 \in [0, 1]$ for $t \in [k-1, k]$. Then, the cubic flow computes the interpolated point and drift as:

$$\theta_t = a_k \tau^3 + b_k \tau^2 + c_k \tau + d_k, \tag{9}$$

$$v_t = 3a_k \tau^2 + 2b_k \tau + c_k, \tag{10}$$

where the coefficients a_k, b_k, c_k, d_k are computed by the four points $\theta_{k-2}, \theta_{k-1}, \theta_k, \theta_{k+1}$ using Catmull-Rom spline [53]. For interpolating the first and the last segment, we set the sentinels as $\theta_{-1} = \theta_0$ and $\theta_{T+1} = \theta_T$. More details are provided in Appendix B.

3.4 Conditional Drift Network Architecture

The conditional drift network h_ϕ consists of a feature extractor, a task projector, and a drift decoder (see Figure 2 for an illustration). Since the target drift v_t of the ODE depends on the task $\mathcal{T}$, we need an effective task encoding mechanism. To this end, we first encode the support set into a task vector $z_\mathcal{T}$ using the feature extractor and the task projector. We first create class prototypes by encoding the support images using a feature extractor and averaging them for each class. For the feature extractor, we use a ResNet-18 [54] pretrained on ImageNet-1k and freeze it during training. It only summarizes the support set into a task condition and does not contribute to query classification. A separate lightweight encoder keeps the condition fixed as the target parameters evolve and its cost constant across target backbones, and it can be replaced by a stronger encoder without changing the objective. Then, a linear task projector transforms the class prototypes to learnable embeddings, which are further averaged to produce a single task vector $z_\mathcal{T}$. Also, the scalar timestep t is encoded by the sinusoidal embedding and linearly projected to a time embedding, which is concatenated to the task vector.

Conditioned on the task vector and time embedding vector, the drift decoder predicts the drift v_t from the vectorized parameters θ_t. The drift decoder is a shallow MLP whose intermediate features are modulated by adaptive layer normalization (AdaLN) modules [55]. In our LoRA-based instantiation as described in Section 5.4, we replace the MLP with a JiT [56] transformer. Each AdaLN module gets the concatenated task vector and time embedding as input, and produces the scale, shift, and gating parameters for each block of the drift decoder. We leave the detailed explanation about the conditioning mechanism of AdaLN and the architectural configurations in Appendix C.

4 Related Work

Few-Shot Learning. Few-shot learning aims to adapt models to handle new tasks with minimal labeled data. A predominant strategy of *few-shot classification*, where each task is defined by

a set of image classes [1, 2], is the metric-based learning that assigns the classes based on the distance on an embedding space learned by a feature extractor [3, 12]. While early methods of this type typically transfer the feature extractor to novel classes without additional adaptation, several studies [14, 15, 16, 17, 18] have shown that test-time fine-tuning offers substantial improvements in cross-domain scenarios. In particular, Hu et al. [18] introduced P>M>F, a three-stage pipeline—self-supervised pretraining, episodic meta-learning, and test-time fine-tuning—that significantly boosts performance on out-of-domain tasks.

Meta-Optimizers and Gradient-Based Meta-Learning. Another branch of few-shot learning trains meta-optimizers or learns good initializations for rapid task adaptation [57, 31, 4]. While these methods reduce reliance on hand-crafted optimizers, they still require gradient-based adaptation. In particular, learned optimizers [57, 4] predict updates from loss gradients and thus still backpropagate through the backbone at adaptation time, whereas HyperFlow uses gradients only offline. LEO [58] adapts a latent code of classifier parameters by gradient descent, and MetaOptNet [59] solves a convex problem for a linear classifier over fixed embeddings. Neither adapts the backbone through learned parameter dynamics. Recent methods replace the inner loop with learned gradient-free mechanisms, such as synthetic gradients [23], Neural ODEs [24], or diffusion models [25], but typically focus on prototypes or linear classifiers and still involve target-model backpropagation during training [19, 20, 21, 22]. For example, MetaNODE [20] evolves class prototypes in the feature space with a Neural ODE under an episodic classification objective, whereas the ODE state of HyperFlow is a vector of backbone PEFT parameters, supervised by the interpolated states and velocities of actual fine-tuning trajectories. HyperFlow extends this direction toward compact PEFT parameter subspaces, with test-time adaptation that does not require backpropagation through the target model.

Hypernetworks and Parameter Generation. Parameter generation methods aim to generate model parameters using another neural network, often called a *hypernetwork* [60]. These methods primarily aim to generate neural network parameters that generalize across various architectures or training configurations rather than adapting to novel tasks with a few examples [27, 29, 26, 28]. Recently, Jin et al. [61] and Liang et al. [62] have proposed few-shot parameter generation methods for natural language processing (and style transfer) and policy learning, respectively. Both rely on latent diffusion models [63] operating in a latent space learned via autoencoders, which often require high computation cost and do not directly focus on efficient adaptation. Our approach, by contrast, learns task-conditioned fine-tuning dynamics in a selected PEFT parameter space, which allows computationally efficient adaptation via a few forward operations of a lightweight drift network. Moreover, while hypernetworks directly generate the final parameters, HyperFlow predicts local drifts and obtains the adapted parameters by ODE solving. We isolate this difference in Section 5.4 by comparing with a hypernetwork baseline that shares the architecture, target parameters, and trajectory data with HyperFlow.

5 Experiments

The main advantage of HyperFlow is its accuracy–efficiency trade-off between direct transfer and gradient-based fine-tuning. As a demonstration, we apply HyperFlow to the P>M>F [18] pipeline, which involves pretraining, meta-training, and fine-tuning stages for cross-domain few-shot classification, and replace its fine-tuning stage with our gradient-free test-time adaptation. Unless otherwise stated, HyperFlow adapts the QKV bias parameters of the backbone (Section 3.2).

5.1 Experimental Setup

Datasets. Following Hu et al. [18], we use the Meta-Dataset benchmark [30], which consists of ten datasets spanning diverse domains: ImageNet-1k (Inet), Omniglot (Omglot), FGVC-Aircraft (Acraft), CUB-200-2011 (CUB), Describable Textures (DTD), QuickDraw (QDraw), FGVCx Fungi (Fungi), VGG Flower (Flower), Traffic Signs (Sign), and MSCOCO (COCO). We adopt the 8 in-domain setup, treating the training splits of the first eight datasets as the *base* dataset and evaluating on the test splits of all ten, where Sign and COCO serve as out-of-domain tasks. Test episodes of the Meta-Dataset tasks are constructed in *various-way various-shot*, where both the number of classes and the number of support images per class are randomly chosen. In addition, we include four out-of-domain datasets from the CD-FSL benchmark [14]: CropDisease, EuroSAT, ISIC, and ChestX. Here, tasks are constructed in 5-way with 5-, 20-, and 50-shots. In all datasets, we evaluate the models with 600 randomly sampled episodes. In all tables, values after $\pm$ denote the standard deviation.

Table 1: Accuracy (%) on Meta-Dataset. HyperFlow provides a gradient-free test-time adaptation alternative that improves over direct transfer while remaining below full fine-tuning in most cases.

Model Variant	Average		Out-of-Domain		In-Domain							
	OOD	ID	Sign	COCO	Acraft	CUB	DTD	Fungi	Flower	Inet	Omglot	QDraw
ViT-S backbone												
Direct Transfer	55.24	83.31	55.35	55.12	87.26	91.40	81.63	74.16	91.06	70.78	90.18	79.98
HyperFlow-L **(Ours)**	59.12	83.42	61.78	56.46	86.76	91.18	82.30	73.83	92.27	70.69	90.15	80.14
HyperFlow-C **(Ours)**	63.11	83.33	69.49	56.72	86.74	91.07	82.05	73.82	91.50	71.48	89.89	80.12
Head-Tuning	55.92	83.74	56.49	55.34	87.21	91.30	83.31	73.80	92.34	71.38	89.96	80.60
Bias-Tuning	74.06	84.69	88.35	59.77	87.49	91.21	86.43	73.88	95.68	72.10	90.62	80.14
LoRA-Tuning	74.97	85.17	90.58	59.36	89.61	91.81	87.12	74.00	94.81	72.08	91.83	80.09
Full-Tuning	75.38	84.91	91.65	59.10	89.32	91.08	87.60	73.80	94.82	70.87	91.93	79.83
ViT-L backbone												
Direct Transfer	62.82	91.17	55.75	69.88	95.73	97.85	82.58	86.21	99.56	90.18	93.98	83.28
HyperFlow-L **(Ours)**	71.68	91.48	71.63	71.72	96.17	97.78	84.08	86.19	99.58	90.70	93.95	83.39
HyperFlow-C **(Ours)**	71.43	91.63	70.43	72.43	95.98	97.97	85.41	86.31	99.60	90.38	93.88	83.50
Head-Tuning	65.16	92.10	60.22	70.11	96.73	97.88	87.77	86.23	99.76	90.23	93.98	84.25
Bias-Tuning	81.02	92.56	89.50	72.54	96.78	97.83	90.75	86.25	99.65	90.98	94.07	84.13
LoRA-Tuning	82.06	92.55	91.77	72.34	97.28	97.88	91.26	86.26	99.61	90.22	94.45	83.41
Full-Tuning	81.71	92.61	91.80	71.63	97.08	97.84	90.43	86.21	99.72	90.79	94.51	84.33

Baselines. We choose the ProtoNet [12] variant proposed by Hu et al. [18] as our target model. The model is first pretrained on ImageNet-1k with a self-supervised objective. In this study, we use a DINO [32]-pretrained ViT-S backbone and a BEiTv2 [64]-pretrained ViT-L backbone. In Appendix A, we also provide results with ConvNeXt-S [65] backbone to show the backbone generality of HyperFlow. Each backbone is then meta-trained on the eight base datasets with episodic training protocol. Using the meta-trained ProtoNet as a base model, we compare three types of approaches throughout the experiments:

- **Direct Transfer** is immediately evaluated without test-time adaptation.
- **Head-Tuning**, **Bias-Tuning**, **LoRA-Tuning**, and **Full-Tuning** fine-tune the parameters of the classifier head (initialized to prototypes), bias, 16-rank LoRA [35], and the whole network, respectively. The fine-tuning baselines involve 50 gradient descent steps with the Adam optimizer [36], and the learning rate is determined by a few validation samples of each task following Hu et al. [18].
- **HyperFlow-L** and **HyperFlow-C** are two representative variants of HyperFlow, replacing the fine-tuning stage of P>M>F pipeline with linear and cubic flows, respectively. For the ODE solving process, we apply the Euler method with at most 50 steps, where the number of steps and the step size are determined similarly to the learning rate selection of the fine-tuning baselines.

Implementation Details. For HyperFlow and the Bias-Tuning variant, we update the bias parameters of QKV-projection from each attention layer of the backbone transformer, which totals 13,824 and 49,152 parameters for ViT-S and ViT-L, respectively. We train the conditional drift network by collecting fine-tuning trajectories from the base ProtoNet model on the eight training domains of the Meta-Dataset. Specifically, we sample 500 episodes per domain and perform 50 gradient descent steps with the Adam optimizer using a fixed learning rate. For each episode, we apply ten different Gaussian perturbations to the initial parameters, yielding a total of 40,000 trajectories. When training HyperFlow-L, we use only the first and final parameters of each trajectory for interpolation. Further details for offline trajectory collection are provided in Appendix D.

5.2 Main Result

Table 1 and Table 2 present the main results on Meta-Dataset and CD-FSL, respectively. Overall, HyperFlow improves over Direct Transfer and remains below gradient-based fine-tuning baselines in most settings, matching our intended accuracy–efficiency trade-off. The gains are most pronounced on out-of-domain (OOD) tasks, especially Sign, COCO, and the CD-FSL domains, where test-time adaptation is more useful. Notably, on the ViT-L backbone in the CD-FSL 5-shot setting, HyperFlow-C achieves 65.57% average accuracy, nearly matching Full-Tuning (66.23%) and LoRA-Tuning (65.95%), while avoiding test-time backpropagation through the target model. These results suggest that the learned drift captures adaptation patterns that transfer beyond the training domains. We examine this interpretation more directly in Appendix E through drift-alignment and shared drift-geometry analyses, which test whether the predicted updates align with fine-tuning directions and whether drift vectors exhibit reusable structure across domains.

Table 2: Accuracy (%) on CD-FSL. HyperFlow generalizes to the OOD tasks with various shots.

Model Variant	Average			EuroSAT			ISIC			CropDisease			ChestX		
	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s
ViT-S backbone															
Direct Transfer	56.53	61.95	63.88	80.56	86.45	87.90	34.89	42.74	45.93	85.80	90.44	91.46	24.88	28.15	30.23
HyperFlow-L (**Ours**)	61.07	66.64	67.70	83.45	88.49	89.60	46.74	55.66	56.24	89.19	92.61	94.22	24.89	29.78	30.73
HyperFlow-C (**Ours**)	62.82	67.24	69.60	86.92	91.25	91.91	46.16	53.31	57.79	93.05	95.30	96.23	25.14	29.09	32.47
Head-Tuning	57.46	63.53	65.96	80.42	85.88	88.93	37.17	46.29	50.52	87.46	92.56	93.52	24.79	29.37	30.86
Bias-Tuning	62.89	71.26	75.43	86.15	93.56	95.81	47.51	61.86	69.39	92.88	97.62	98.80	25.01	32.01	37.72
LoRA-Tuning	64.25	72.90	77.19	86.18	93.93	96.64	50.10	65.19	72.88	93.58	98.26	99.23	27.12	34.23	40.00
Full-Tuning	64.76	72.95	76.87	87.10	94.62	96.56	50.38	64.87	72.49	94.69	98.38	99.26	26.86	33.93	39.17
ViT-L backbone															
Direct Transfer	59.32	64.30	65.95	75.76	82.82	84.46	41.92	49.00	51.79	95.48	97.00	97.22	24.13	28.37	30.32
HyperFlow-L (**Ours**)	63.24	68.18	69.51	81.50	86.56	87.41	51.29	60.46	62.69	95.40	97.09	97.44	24.77	28.61	30.50
HyperFlow-C (**Ours**)	65.57	70.41	72.23	86.62	91.01	92.00	54.03	63.21	66.22	96.10	97.51	97.82	25.51	29.90	32.87
Head-Tuning	63.05	69.39	71.59	81.93	90.31	92.37	49.38	60.60	64.67	95.46	98.10	98.30	25.40	28.52	31.01
Bias-Tuning	65.37	72.88	76.47	86.44	94.73	96.79	52.97	66.36	73.01	96.20	98.96	99.39	25.88	31.48	36.68
LoRA-Tuning	65.95	73.72	77.77	87.37	94.86	97.01	53.40	67.80	75.31	96.61	98.94	99.48	26.43	33.26	39.27
Full-Tuning	66.23	73.61	78.15	87.87	95.25	97.28	53.08	67.86	75.10	97.26	99.14	99.60	26.72	32.20	40.63

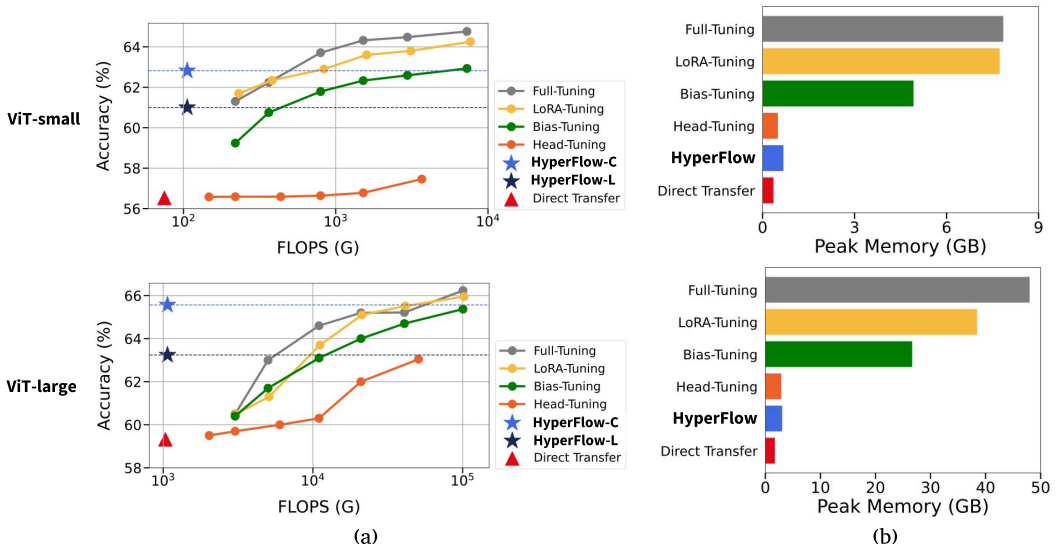


Figure 3: Computation cost analysis of HyperFlow in P>M>F framework. (a) Average 5-shot CD-FSL accuracy over FLOPs (log scale). HyperFlow occupies an intermediate accuracy–efficiency regime between direct transfer and fine-tuning. (b) Peak memory requirement in a 40-way 5-shot setting. HyperFlow substantially reduces memory usage compared with Bias-Tuning and Full-Tuning, while adding a moderate overhead over Direct Transfer.

Compared with Head-Tuning, HyperFlow-C generally achieves stronger OOD performance while requiring substantially lower FLOPs and comparable peak memory, as shown in Figure 3. Comparing the two HyperFlow variants, HyperFlow-C performs better than HyperFlow-L on most OOD tasks, suggesting that the linear endpoint interpolation can oversimplify the fine-tuning trajectory. We further analyze the effect of the ODE objective in Section 5.4.

5.3 Computation Cost Analysis

In this section, we provide a detailed analysis of the memory and compute efficiency of HyperFlow. Further training cost analysis and a comprehensive summary are provided in Appendix E and G.

Compute Efficiency. Figure 3-(a) shows the average accuracy of CD-FSL 5-shot tasks over FLOPs (floating-point operations) of each method for a single test episode. By decoupling the adaptation process from the target model, HyperFlow requires merely 1% of FLOPs of the Full-Tuning and Bias-Tuning approaches, translating to only $0.4\times$ increase over Direct Transfer. Therefore,

Table 3: Results of HyperFlow applied to TSA framework.

Model Variants	Meta-Dataset Average		CD-FSL Average		
	OOD	ID	5w5s	5w20s	5w50s
TSA w/o Fine-Tuning	33.58 ± 0.91	65.21 ± 0.89	58.62 ± 0.56	63.68 ± 0.46	65.60 ± 0.45
HyperFlow-C	35.55 ± 1.02	69.29 ± 0.84	59.22 ± 0.56	64.09 ± 0.47	65.87 ± 0.46
TSA	41.42 ± 1.02	68.67 ± 0.83	60.15 ± 0.55	65.92 ± 0.51	68.94 ± 0.51

Table 4: Results of HyperFlow applied to rank-4 LoRA parameters.

Model Variants	Meta-Dataset Average		CD-FSL Average		
	OOD	ID	5w5s	5w20s	5w50s
Direct Transfer	55.24 ± 1.04	83.31 ± 0.60	56.53 ± 0.54	61.95 ± 0.48	63.88 ± 0.45
HyperFlow-C	56.56 ± 1.03	83.42 ± 0.60	63.20 ± 0.49	68.12 ± 0.40	69.93 ± 0.39
LoRA-Tuning	74.97 ± 0.82	85.17 ± 0.60	64.25 ± 0.52	72.90 ± 0.38	77.19 ± 0.35

HyperFlow occupies the middle ground between direct-transfer and fine-tuning in both performance and computation cost. It is noticeable that HyperFlow is even more efficient than the single-step Head-Tuning (the leftmost point on the orange line) while achieving much higher accuracy. These results suggest that HyperFlow can be highly advantageous in the real-world edge devices that demand cheap and fast computation for test-time adaptation.

Memory Efficiency. Figure 3-(b) reports the peak memory usage of each method for a single 40-way, 5-shot episode with 128×128 images. HyperFlow uses only 6–14% of the peak memory required by gradient-based fine-tuning methods, while introducing less than a $1\times$ additional memory overhead over Direct Transfer. This reduction comes from avoiding activation storage for target-model backpropagation during test-time adaptation. As a result, HyperFlow is particularly appealing for adapting large-scale models in low-resource environments, where fine-tuning would be infeasible due to the memory constraint. For example, on ViT-L, HyperFlow requires 3.13 GB compared with 27.31 GB for Bias-Tuning, an $8.7\times$ reduction ($7.3\times$ on ViT-S), fitting within an 8–16 GB budget where Bias-Tuning does not (Appendix G). Thus, HyperFlow targets the regime where adaptation is beneficial but test-time backpropagation is unaffordable.

Offline Cost. The offline stage is performed once per backbone and PEFT configuration and reused across unseen tasks. With 40K trajectories on ViT-S, trajectory collection and drift training take 9.6 h and 0.5 h, adding about 27% wall-clock time and 11% storage to the 37.5 h meta-training (Appendix E). With 400 trajectories, the overhead drops to 1.6% and 0.11%, while HyperFlow still improves Direct Transfer from 55.24% to 60.64% (Meta-Dataset OOD) and from 56.53% to 62.12% (CD-FSL 5-shot).

5.4 Ablation Study

We conduct ablation studies of HyperFlow using ViT-S, unless explicitly mentioned otherwise. More analyses and results are included in Appendix E and F.

Generalization in Other Few-Shot Adaptation Framework. To examine whether HyperFlow is tied to the P>M>F pipeline used in our main experiments, we additionally instantiate it on TSA [50], a representative cross-domain few-shot adaptation framework that fine-tunes task-specific adapters at test time. We use the task-specific adapters as target parameters for HyperFlow; the remaining details are the same as in the main experiment. As shown in Table 3, HyperFlow again yields a consistent improvement over direct transfer while remaining below or close to test-time fine-tuning, following the same accuracy–efficiency pattern observed in the main experiments. This provides preliminary evidence that HyperFlow is not tied only to the ProtoNet-based P>M>F setup, but can be instantiated in other few-shot frameworks by training a corresponding drift network for the relevant parameters.

Scaling to LoRA Parameter Space. To examine whether HyperFlow scales beyond the compact bias subspace, we apply HyperFlow-C to rank-4 LoRA [35] on the QKV projections of ViT-S, whose adaptation space is $5.3\times$ larger than that of the QKV bias. We collect LoRA-tuning trajectories and replace the MLP-based drift decoder with a JiT [56] backbone, while keeping the training objective and the ODE-based adaptation procedure unchanged. As shown in Table 4, HyperFlow improves over Direct Transfer in the LoRA space as well. Compared with the default bias parameterization (Table 2), it achieves higher CD-FSL accuracy in all shot settings and slightly exceeds Bias-Tuning in the 5-shot setting. Its peak memory increases only to about 11% of that of Full-Tuning (8.6%

Table 5: Ablation study on the ODE objectives of HyperFlow.

Model Variants	Meta-Dataset Average		CD-FSL Average		
	OOD	ID	5w5s	5w20s	5w50s
HyperNet	59.51 ± 1.01	83.56 ± 0.61	59.70 ± 0.49	64.93 ± 0.44	67.05 ± 0.41
HyperFlow-L	59.12 ± 1.01	83.42 ± 0.62	61.07 ± 0.53	66.64 ± 0.47	67.70 ± 0.43
HyperFlow-PL	61.59 ± 0.98	83.34 ± 0.63	62.62 ± 0.48	67.39 ± 0.41	69.60 ± 0.39
HyperFlow-D	61.47 ± 0.97	83.32 ± 0.63	62.19 ± 0.48	67.29 ± 0.41	69.23 ± 0.40
HyperFlow-C	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39

Table 6: Ablation study on the target parameters of HyperFlow.

Model Variants	Meta-Dataset Average		CD-FSL Average		
	OOD	ID	5w5s	5w20s	5w50s
QKV Bias	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39
MLP Bias	57.08 ± 1.01	83.22 ± 0.63	59.48 ± 0.51	63.95 ± 0.45	65.64 ± 0.43
QKV&MLP Bias	58.57 ± 1.00	83.25 ± 0.63	58.41 ± 0.54	64.66 ± 0.44	66.40 ± 0.42

with bias parameters), showing that HyperFlow can exploit a more expressive PEFT space while preserving its memory advantage.

Effect of ODE-based Inference. To investigate the impact of the ODE-based adaptation mechanism, we compare several variants of HyperFlow, including two non-ODE variants (**HyperNet** and **HyperFlow-D**), as well as the piecewise-linear variant (**HyperFlow-PL**) introduced in Section 3.3. All methods share the same training trajectories and architecture, but the non-ODE variants directly predict the next parameters rather than solving an ODE. Specifically, HyperNet predicts the final parameters θ_T given initial parameters θ_0 in one step, while HyperFlow-D iteratively updates the parameters similar to the gradient descent. They can be seen as simplified versions of HyperFlow-L and HyperFlow-PL, where they only predict the drift at the discretized timesteps. Note that HyperNet is a natural generalization of the original hypernetworks proposed by Ha et al. [60] for few-shot learning, using our implementation of the conditional drift network and target parameters.

Table 5 summarizes their performance on the Meta-Dataset and CD-FSL benchmarks. First, we observe that exploiting intermediate trajectories is beneficial in OOD generalization, where the nonlinear variants (HyperFlow-D, HyperFlow-PL, and HyperFlow-C) consistently outperform the linear variants (HyperNet and HyperFlow-L). Also, HyperFlow-L outperforms HyperNet in CD-FSL tasks, which indicates that iterative updates at test-time adaptation can help OOD generalization. Finally, we observe that HyperFlow-C generally performs better than the non-smooth variants (HyperFlow-D and HyperFlow-PL), which shows the robustness of the smooth ODE interpolation.

Choice of Target Parameters. HyperFlow requires selecting a compact parameter subspace whose adaptation dynamics are modeled by the drift network. To study the sensitivity to this design choice, we compare several bias-parameter subsets, including QKV bias, MLP bias, and their combination. As shown in Table 6, all tested parameter choices improve over Direct Transfer on the OOD averages, indicating that HyperFlow is not limited to a single hand-picked subspace. However, the choice of subspace clearly matters: MLP-related subspaces are less effective than QKV bias and can degrade in-domain performance. This suggests that the effectiveness of HyperFlow depends not only on the number of adapted parameters, but also on whether the selected subspace provides transferable and learnable adaptation dynamics.

6 Conclusion

We introduced HyperFlow, a gradient-free test-time adaptation method that emulates fine-tuning dynamics in a compact PEFT parameter subspace. By viewing fine-tuning trajectories as discretized ODE paths, HyperFlow trains a conditional drift network from offline trajectories and performs test-time adaptation using only forward passes of the drift network. Across cross-domain few-shot classification benchmarks, HyperFlow improves over direct transfer while requiring substantially less memory and FLOPs than gradient-based fine-tuning. Additional analyses suggest that HyperFlow learns reusable adaptation structure in the selected PEFT subspace. A current limitation is that HyperFlow still often remains below gradient-based fine-tuning baselines in raw accuracy, so it is not intended to fully replace fine-tuning when deployment-time backpropagation is affordable. Overall, HyperFlow provides a practical intermediate point between direct transfer and gradient-based fine-tuning for settings where test-time backpropagation is constrained.

Acknowledgments and Disclosure of Funding

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Ministry of Science and ICT (MSIT) of the Republic of Korea (RS-2024-00351212, RS-2024-00436165, RS-2024-00509279, RS-2022-II220926, RS-2022-II220959, and RS-2026-25518317). This work was also supported by the InnoCORE program of the MSIT (AI Meta-Scientist, N10260110).

References

[1] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[2] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[3] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[4] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[5] Jai Kotia, Adit Kotwal, Rishika Bharti, and Ramchandra Mangrulkar. Few shot learning for medical imaging. In *Machine Learning Algorithms for Industrial Applications*, pages 107–132. Springer, 2020.

[6] David Argüeso, Artzai Picon, Unai Irusta, Alfonso Medela, Miguel G San-Emeterio, Arantza Bereciartua, and Aitor Alvarez-Gila. Few-shot learning approach for plant disease classification using images taken in the field. *Computers and Electronics in Agriculture*, 175:105542, 2020.

[7] Xian Sun, Bing Wang, Zhirui Wang, Hao Li, Hengchao Li, and Kun Fu. Research progress on few-shot learning for remote sensing image interpretation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:2387–2402, 2021.

[8] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[9] Fan Liu, Tianshu Zhang, Wenwen Dai, Chuanyi Zhang, Wenwen Cai, Xiaocong Zhou, and Delong Chen. Few-shot adaptation of multi-modal foundation models: A survey. *Artificial Intelligence Review*, 57(10):268, 2024.

[10] Zhuoyan Xu, Zhenmei Shi, Junyi Wei, Fangzhou Mu, Yin Li, and Yingyu Liang. Towards few-shot adaptation of foundation models via multitask finetuning. In *The Twelfth International Conference on Learning Representations*, 2024.

[11] Anish Madan, Neehar Peri, Shu Kong, and Deva Ramanan. Revisiting few-shot object detection with vision-language models. *Advances in Neural Information Processing Systems*, 37:19547–19560, 2024.

[12] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[13] Peyman Bateni, Raghav Goyal, Vaden Masrani, Frank Wood, and Leonid Sigal. Improved few-shot visual classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14493–14502, 2020.

[14] Yunhui Guo, Noel C Codella, Leonid Karlinsky, James V Codella, John R Smith, Kate Saenko, Tajana Rosing, and Rogerio Feris. A broader study of cross-domain few-shot learning. In *European Conference on Computer Vision*, pages 124–141. Springer, 2020.

[15] Cheng Perng Phoo and Bharath Hariharan. Self-training for few-shot transfer across extreme task differences. In *International Conference on Learning Representations*, 2021.

[16] Ashraful Islam, Chun-Fu Richard Chen, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, and Richard J Radke. Dynamic distillation network for cross-domain few-shot recognition with unlabeled data. *Advances in Neural Information Processing Systems*, 34:3584–3595, 2021.

[17] Jaehoon Oh, Sungnyun Kim, Namgyu Ho, Jin-Hwa Kim, Hwanjun Song, and Se-Young Yun. Understanding cross-domain few-shot learning based on domain similarity and few-shot difficulty. *Advances in Neural Information Processing Systems*, 35:2622–2636, 2022.

[18] Shell Xu Hu, Da Li, Jan Stühmer, Minyoung Kim, and Timothy M Hospedales. Pushing the limits of simple pipelines for few-shot learning: External data and fine-tuning make a difference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9068–9077, 2022.

[19] Shell Xu Hu, Pablo Garcia Moreno, Yang Xiao, Xi Shen, Guillaume Obozinski, Neil Lawrence, and Andreas Damianou. Empirical Bayes transductive meta-learning with synthetic gradients. In *International Conference on Learning Representations*, 2020.

[20] Baoquan Zhang, Xutao Li, Shanshan Feng, Yunming Ye, and Rui Ye. MetaNODE: Prototype optimization as a neural ODE for few-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9014–9021, 2022.

[21] Yingjun Du, Zehao Xiao, Shengcai Liao, and Cees Snoek. ProtoDiff: Learning to learn prototypical networks by task-guided diffusion. *Advances in Neural Information Processing Systems*, 36:46304–46322, 2023.

[22] Baoquan Zhang, Chuyao Luo, Demin Yu, Xutao Li, Huiwei Lin, Yunming Ye, and Bowen Zhang. MetaDiff: Meta-learning with conditional diffusion for few-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16687–16695, 2024.

[23] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning*, pages 1627–1635. PMLR, 2017.

[24] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

[25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[26] Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 34:29433–29448, 2021.

[27] William S Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *CoRR*, 2022.

[28] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35:27906–27920, 2022.

[29] Konstantin Schürholt, Michael W Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Forty-first International Conference on Machine Learning*, 2024.

[30] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-Dataset: A dataset of datasets for learning to learn from few examples. In *International Conference on Learning Representations*, 2020.

[31] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[32] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9650–9660, 2021.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[34] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, 2022.

[35] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *ICLR (Poster)*, 2015.

[37] Daniil Merkulov and Ivan Oseledets. Stochastic gradient algorithms from ODE splitting perspective. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[38] Omer Elkabetz and Nadav Cohen. Continuous vs. discrete optimization of deep neural networks. *Advances in Neural Information Processing Systems*, 34:4947–4960, 2021.

[39] Zhiqi Bu, Shiyun Xu, and Kan Chen. A dynamical view on optimization algorithms of overparameterized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 3187–3195. PMLR, 2021.

[40] Filippo Santambrogio. Euclidean, metric, and Wasserstein gradient flows: an overview. *Bulletin of Mathematical Sciences*, 7:87–154, 2017.

[41] 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, 2022.

[42] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023.

[43] Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. In *Advances in Neural Information Processing Systems*, 2023.

[44] Zhanpeng Zhou, Zijun Chen, Yilan Chen, Bo Zhang, and Junchi Yan. On the emergence of cross-task linearity in pretraining-finetuning paradigm. In *International Conference on Machine Learning*, pages 61854–61884, 2024.

[45] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[46] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *Advances in Neural Information Processing Systems*, 30, 2017.

[47] Donggyun Kim, Jinwoo Kim, Seongwoong Cho, Chong Luo, and Seunghoon Hong. Universal few-shot learning of dense prediction tasks with visual token matching. In *The Eleventh International Conference on Learning Representations*, 2023.

[48] Donggyun Kim, Seongwoong Cho, Semin Kim, Chong Luo, and Seunghoon Hong. Chameleon: A data-efficient generalist for dense visual prediction in the wild. In *European Conference on Computer Vision*, pages 422–441. Springer, 2024.

[49] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. In *The Eleventh International Conference on Learning Representations*, 2023.

[50] Wei-Hong Li, Xialei Liu, and Hakan Bilen. Cross-domain few-shot learning with task-specific adapters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2022.

[51] C Hermite. Remarque sur le développement de cosamx. *Comptes Rendus de l'Académie des Sciences*, 57:613–618, 1863.

[52] Carl De Boor. *A practical guide to splines*, volume 27. Springer New York, 1978.

[53] Edwin Catmull and Raphael Rom. A class of local interpolating splines. In *Computer Aided Geometric Design*, pages 317–326. Elsevier, 1974.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[55] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[56] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 36115–36125, 2026.

[57] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3988–3996, 2016.

[58] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

[59] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[60] David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[61] Xiaolong Jin, Kai Wang, Dongwen Tang, Wangbo Zhao, Yukun Zhou, Junshu Tang, and Yang You. Conditional LoRA parameter generation. *arXiv preprint arXiv:2408.01415*, 2024.

[62] Yongyuan Liang, Tingqiang Xu, Kaizhe Hu, Guangqi Jiang, Furong Huang, and Huazhe Xu. Make-An-Agent: A generalizable policy network generator with behavior-prompted diffusion. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[63] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[64] Zhiliang Peng, Li Dong, Hangbo Bao, Qixiang Ye, and Furu Wei. BEiT v2: Masked image modeling with vector-quantized visual tokenizers. *arXiv preprint arXiv:2208.06366*, 2022.

[65] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[66] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[67] Adrian El Baz, Ihsan Ullah, Edesio Alcobaça, André C. P. L. F. Carvalho, Hong Chen, Fabio Ferreira, Henry Gouk, Chaoyu Guan, Isabelle Guyon, Timothy Hospedales, Shell Hu, Mike Huisman, Frank Hutter, Zhengying Liu, Felix Mohr, Ekrem Öztürk, Jan N. van Rijn, Haozhe Sun, Xin Wang, and Wenwu Zhu. Lessons learned from the NeurIPS 2021 MetaDL challenge: Backbone fine-tuning without episodic meta-learning dominates for few-shot learning image classification. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 80–96, 2022.

[68] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763, 2021.

Appendix

A Additional Results with ConvNeXt Backbone

In this section, we evaluate HyperFlow on a ConvNeXt-S backbone to test whether the same trend also appears on a convolutional architecture. In this backbone, we choose the target parameters to be the bias parameters of convolution layers of the 5-th stage, which consists of 27 convolution layers, resulting in 10,368 parameters in total. Table 7 and 8 show the quantitative results of the ConvNeXt backbone, where Figure 4 shows the corresponding computation cost analysis. We observe that the overall trends observed from the ViT backbones are preserved, where HyperFlow establishes a practical middle ground between Direct Transfer and fine-tuning baselines. This result suggests that HyperFlow can also be instantiated on convolutional architectures.

Table 7: Few-shot classification accuracy (%) on Meta-Dataset with ConvNeXt-S backbone.

Model Variant	Average		Out-of-Domain		In-Domain							
	OOD	ID	Sign	COCO	Acraft	CUB	DTD	Fungi	Flower	Inet	Omglot	QDraw
Direct Transfer	56.82	87.91	56.17	57.47	91.78	92.93	81.65	74.97	96.40	91.78	92.93	80.87
HyperFlow-C **(Ours)**	60.05	88.09	62.51	57.59	92.90	92.93	81.65	74.97	96.40	92.22	92.78	80.87
Head-Tuning	56.83	88.56	56.17	57.48	92.89	92.92	85.02	74.97	97.13	91.78	92.92	80.87
Bias-Tuning	73.35	88.85	86.78	59.92	92.99	93.33	86.05	74.98	97.10	92.06	93.05	81.26
Full-Tuning	74.53	89.27	90.19	58.87	93.84	93.35	88.46	74.87	97.18	92.03	93.18	81.27

Table 8: Few-shot classification accuracy (%) on CD-FSL with ConvNeXt-S backbone.

Model Variant	Average			EuroSAT			ISIC			CropDisease			ChestX		
	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s
Direct Transfer	57.08	63.00	65.46	77.58	85.32	87.01	37.82	46.32	50.35	87.89	91.42	92.55	25.04	28.92	31.91
HyperFlow-C **(Ours)**	60.08	64.60	66.54	84.84	86.81	87.61	41.60	50.33	53.57	88.83	92.33	93.30	25.04	28.91	31.67
Head-Tuning	60.98	67.98	71.09	82.97	90.24	92.58	42.99	55.35	60.79	92.17	96.14	97.2	25.77	30.19	33.77
Bias-Tuning	62.41	71.29	75.75	85.26	93.29	95.81	45.81	61.98	70.53	92.74	97.72	98.79	25.82	32.16	37.87
Full-Tuning	64.47	73.19	77.46	86.13	94.30	96.58	49.97	65.39	73.50	94.62	98.49	99.26	27.17	34.58	40.50



(a) (b)

Figure 4: Computation cost analysis with ConvNeXt-S backbone. (a) Average 5-shot CD-FSL accuracy over FLOPs (plotted on a log scale). HyperFlow offers an effective middle ground between direct-transfer and fine-tuning. (b) Peak memory requirement in a 40-way 5-shot setting.

B Details about Cubic Flow

In this section, we explain the cubic flow objective introduced in Section 3.3 in detail. From the gradient simulation, each trajectory consists of a sequence of points $\{\theta_k\}_{k=0}^{T}$. The objective of cubic flow is to apply cubic splines on the points by treating $\theta(t)$ as a function of time where $\theta(k) = \theta_k, \forall k = 0, \ldots, T$. Specifically, after dividing the time interval $[0, T]$ into T segments, we interpolate each segment $[k-1, k]$ with a cubic Hermite spline [51, 52] that requires the two endpoints θ_{k-1}, θ_k and their tangents m_{k-1}, m_k to compute the coefficients of the cubic curve. Since we uniformly divided the time interval, we can employ the Catmull-Rom spline [53] that sets the tangents using two additional points $\theta_{k-2}, \theta_{k+1}$. Let $\tau = t - k + 1 \in [0, 1]$ for $t \in [k-1, k]$. Then, the point θ_t and its drift v_t in the segment $[k-1, k]$ can be computed as follows:

$$\theta_t = a_k \tau^3 + b_k \tau^2 + c_k \tau + d_k, \tag{11}$$

$$v_t = 3a_k \tau^2 + 2b_k \tau + c_k, \tag{12}$$

where the coefficients a_k, b_k, c_k, d_k are computed from the four points $\theta_{k-2}, \theta_{k-1}, \theta_k, \theta_{k+1}$ as follows:

$$a_k = -\frac{1}{2}\theta_{k-2} + \frac{3}{2}\theta_{k-1} - \frac{3}{2}\theta_k + \frac{1}{2}\theta_{k+1} \tag{13}$$

$$b_k = \theta_{k-2} - \frac{5}{2}\theta_{k-1} + 2\theta_k - \frac{1}{2}\theta_{k+1} \tag{14}$$

$$c_k = -\frac{1}{2}\theta_{k-2} + \frac{1}{2}\theta_k \tag{15}$$

$$d_k = \theta_{k-1} \tag{16}$$

For interpolating the first and the last segment, we set the sentinels as $\theta_{-1} = \theta_0$ and $\theta_{T+1} = \theta_T$.

C Details about the Drift Decoder

In this section, we provide detailed explanations about the drift decoder and its architecture.

MLP-based Drift Decoder for Bias and TSA-Adapter. As introduced in Section 3.4, the drift decoder is a shallow MLP whose intermediate features are modulated by adaptive layer normalization [55] (AdaLN) modules. The drift decoder consists of a linear input layer, L conditional MLP blocks, and a linear output layer. First, the input layer transforms the flattened parameters $\theta_t \in \mathbb{R}^n$ into a latent vector $\mathbf{p}^{(0)} \in \mathbb{R}^d$, where n and d are the number of bias parameters and the hidden dimension, respectively. Then, each conditional MLP block at layer l gets the latent features from the previous layer $\mathbf{p}^{(l-1)}$ and a condition vector $\mathbf{c}^{(l)}$, which is the concatenation of the task vector $z_{\mathcal{T}}$ from support encoder and the time embedding vector, to produce output features $\mathbf{p}^{(l)}$ as follows:

$$\mathbf{p}^{(l)} = \mathbf{p}^{(l-1)} + \alpha(\mathbf{c}^{(l)}) \cdot MLP(\gamma(\mathbf{c}^{(l)}) \cdot LN(\mathbf{p}^{(l-1)}) + \beta(\mathbf{c}^{(l)})), \tag{17}$$

where MLP is a 2-layer MLP and α, β, γ are the AdaLN layers that produce gating, shift, and scaling parameters, respectively. Each AdaLN layer consists of the SiLU activation [66] and a linear layer. Finally, the output layer transforms the last latent features $p^{(L)} \in \mathbb{R}^d$ into the drift $v_t \in \mathbb{R}^n$.

In Table 9, we provide the architectural configuration of the drift decoder used in our experiments. All the variants (HyperNet, HyperFlow-L, HyperFlow-C, HyperFlow-D, and HyperFlow-PL) follow the same architecture.

JiT-based Drift Decoder for LoRA. For the LoRA parameterization (Section 5.4), we replace the MLP-based drift decoder with a Transformer following JiT [56]. Each block consists of RMSNorm, multi-head self-attention with query-key normalization and 1D rotary position embedding, and a SwiGLU feed-forward layer, and its features are modulated by AdaLN conditioned on the task vector and the time embedding. We tokenize the rank-4 QKV LoRA parameters at the granularity of rank components, where each token concatenates a row of the down-projection matrix and the corresponding column of the up-projection matrix in an attention layer. For ViT-S, this yields $12 \times 4 = 48$ tokens of dimension $384 + 1{,}152 = 1{,}536$, which are embedded by a linear bottleneck ($1{,}536 \to 128 \to 512$) with fixed sinusoidal position embeddings. The decoder consists of 12 blocks with a hidden dimension of 512, 8 attention heads, and an MLP ratio of 4. After the first four blocks, the class prototypes of the support set are appended to the parameter tokens as in-context tokens, one token per class.

Other differences from the default bias version are as follows. First, the LoRA parameters are standardized with the per-coordinate mean and standard deviation computed from the training trajectories before being fed to the decoder. Second, instead of predicting the drift directly, the decoder predicts the endpoint parameters $\hat{\theta}_T$, which are converted to the drift as $(\hat{\theta}_T - \theta_t)/\max(1 - t, 0.05)$ with the time normalized to $[0, 1]$, while the training loss is computed on the drift as in Eq. (4). Third, the LoRA-tuning trajectories are collected with a learning rate of 0.005 and a noise level of $\sigma = 0.1$, instead of 0.05 and 0.2 used for the bias parameters of ViT-S (Table 10).

D Details about the Trajectory Collection

During the offline trajectory collection, we use Adam [36] optimizer to fine-tune the bias parameters of the base model. Similar to the episodic meta-training, we randomly sample classification tasks from

Table 9: Architectural configuration of the drift decoder of HyperFlow.

hyperparameter	ViT-S	ViT-L	ConvNeXt-S
# of bias parameters (n)	13,824	49,152	10,368
hidden dimension (d)	384	384	384
# of conditional MLP blocks (L)	1	1	1

Table 10: Learning rates and noise levels used in offline trajectory collection.

Backbone	ViT-S	ViT-L	ConvNeXt-S
Learning rate	0.05	0.02	0.002
Noise level (σ)	0.2	0.2	0.015

Table 11: Drift alignment analysis. We report the cosine similarity between the drift predicted by HyperFlow and the reference fine-tuning drift. Positive values indicate that the predicted drift is aligned with the gradient-based adaptation direction.

Trajectory Source	Domain	Steps					Average
		1-10	11-20	21-30	31-40	41-50	
Fine-Tuning	ID mean	0.4435	0.3496	0.2656	0.2111	0.1652	0.2870
	OOD mean	0.2040	0.1842	0.1543	0.1250	0.0878	0.1510
	All mean	0.3409	0.2787	0.2179	0.1742	0.1320	0.2287
HyperFlow	ID mean	0.4120	0.2997	0.2122	0.1612	0.1244	0.2419
	OOD mean	0.1176	0.0906	0.0609	0.0462	0.0327	0.0696
	All mean	0.2858	0.2101	0.1473	0.1119	0.0851	0.1681

Table 12: Shared drift geometry analysis. We report the explained variance ratio (EVR) of ground-truth fine-tuning drift vectors under different PCA bases. "Domain" denotes a domain-specific basis fitted on the target domain, "Pooled" denotes a basis fitted on all domains, and "LOO" denotes a leave-one-domain-out basis fitted without the target domain.

Rank k	Domain	Pooled	LOO	Pooled / Domain
1	0.11	0.03	0.02	29.9%
5	0.22	0.08	0.04	37.4%
10	0.34	0.12	0.05	34.5%
20	0.54	0.16	0.07	30.3%
50	0.78	0.27	0.10	34.1%
ID mean (k=10)	0.28	0.11	0.06	40.3%
OOD mean (k=10)	0.45	0.12	0.04	27.5%
Random (k=10)	–	–	0.0007	–

the base dataset and fine-tune the parameters with these tasks. Also, we add a random perturbation $\epsilon \sim \mathcal{N}(0, \sigma^2)$ to the initialization of the parameters. Table 10 summarizes the learning rates and initial noise used in the offline trajectory collection of various backbones.

E Additional Analysis

Drift Alignment Analysis. To directly examine whether HyperFlow learns fine-tuning-like update directions, we measure the cosine similarity between the drift predicted by the conditional drift network and the corresponding gradient-based fine-tuning direction. Given a trajectory point θ_k, we define the reference drift as the optimizer-induced update direction obtained by one fine-tuning step on the support set, and compare it with the predicted drift $h_\phi(\theta_k, k \mid \mathcal{S}_\mathcal{T})$. We evaluate this alignment under two trajectory sources: points sampled from actual fine-tuning trajectories and points sampled from the trajectories generated by HyperFlow itself. As shown in Table 11, the predicted drift has consistently positive alignment with the fine-tuning direction, with average cosine similarities of 0.2287 along fine-tuning trajectories and 0.1681 along HyperFlow trajectories. The

Table 13: Wall-clock time and storage required by each training stage of HyperFlow (ViT-S).

	Meta-training	Offline trajectory collection						Drift network training
		HyperFlow-L			HyperFlow-C			
		400 traj	4k traj	40k traj	400 traj	4k traj	40k traj	
Wall-clock time (hour)	37.5	0.096	0.96	9.6	0.096	0.96	9.6	0.5
Storage (GB)	481	0.021	0.21	2.1	0.53	5.3	53	-

alignment is stronger on in-domain tasks and decreases over later steps, which is expected because the update magnitude becomes smaller and the trajectories can diverge from the supervised fine-tuning path. These positive alignment values suggest that HyperFlow is not only predicting endpoint-like parameter offsets, but also partially tracks the local adaptation directions induced by fine-tuning.

Shared Drift Geometry Analysis. We further analyze whether the fine-tuning drifts contain reusable structure across domains. For each domain, we collect ground-truth fine-tuning drift vectors and compute the explained variance ratio (EVR) of their principal components. We compare three bases: a domain-specific PCA basis fitted on the target domain, a pooled basis fitted on drift vectors from all domains, and a leave-one-domain-out basis fitted on all domains except the target domain. The domain-specific basis represents an upper bound for low-rank reconstruction within each domain, while the pooled and leave-one-out bases measure how much drift structure is shared across domains. As reported in Table 12, the pooled rank-k basis explains a stable fraction of the variance captured by the domain-specific basis, with a pooled/domain-specific ratio of approximately 30–37% for $k \in \{1, 5, 10, 20, 50\}$. Moreover, the leave-one-domain-out basis explains substantially more drift energy than a random basis; for example, at $k = 10$, the leave-one-out basis explains about 5% of the drift variance, compared to only 0.07% for a random basis. These results suggest that fine-tuning dynamics in the selected PEFT subspace contain a reusable low-dimensional component across domains. We therefore interpret the generalization of HyperFlow as arising partly from learning this reusable adaptation structure, rather than from memorizing domain-specific trajectories.

HyperFlow Training Overhead. Compared to the standard P>M>F pipeline, HyperFlow adds two offline stages: collecting fine-tuning trajectories and training the drift network. These stages incur extra time and storage overhead, but they are designed to be performed *once* on the server side, where larger computational resources are typically available, analogous to the pre-training and meta-training stages. Importantly, the overhead of HyperFlow training is modest relative to meta-training and can be made very small in practice. We report the wall-clock time and storage required by each stage using eight NVIDIA A6000 GPUs. As summarized in Table 13, collecting 40k full trajectories of bias parameters and training the drift network introduces only about 27% additional training time and 11% additional storage compared to meta-training. Moreover, as we show in Table 14, HyperFlow can still achieve substantial gains over Direct Transfer with a much smaller trajectory budget (*e.g.,* 400 trajectories), under which the offline overhead becomes almost negligible (1.6% additional time and 0.11% additional storage). These trajectories are used only during the offline drift training and are not stored on deployment devices.

Intermediate Loss Curves Induced by HyperFlow. To further investigate the OOD generalization of HyperFlow, we plot the relative losses to the initial loss on CD-FSL 5-shot tasks during the ODE solving process of HyperFlow-C in Figure 5. In general, HyperFlow moves the parameters into regions with lower loss values, indicating that it effectively generalizes to OOD tasks. We observe that the loss does not drastically decrease on ChestX tasks, which can be attributed to the inherent limitation of the meta-trained ProtoNet backbone, since the fine-tuning baselines also show marginal improvement over Direct Transfer.

Effect of Trajectory Diversity and Construction. To further examine where the generalization ability of HyperFlow originates, we conduct ablation studies that vary both the diversity and the construction of the offline training trajectories. Table 14 summarizes the results when HyperFlow-C is trained with different numbers of domains, episodes, initializations, fine-tuning steps, and trajectory-generation optimizers. To analyze domain diversity, we train the drift network using trajectories from a single domain (Inet), four domains (Inet, Omglot, Acraft, CUB), and all eight meta-training domains. We also vary the number of sampled episodes and the number of random initializations used

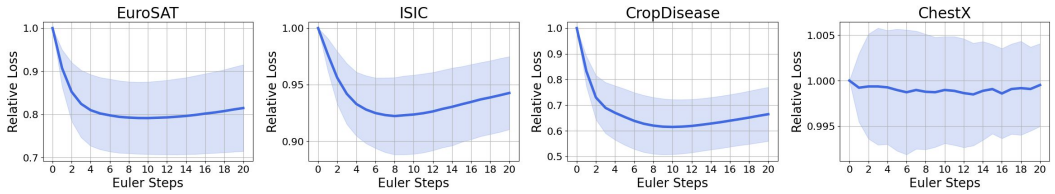
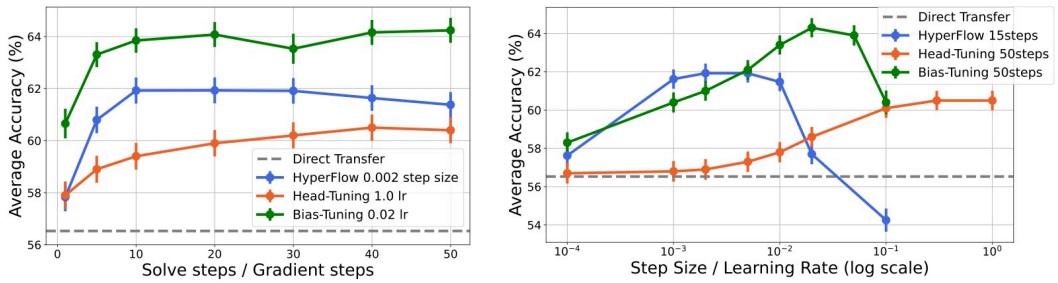


Figure 5: Relative losses on CD-FSL tasks during the ODE solving process of HyperFlow-C. We plot the average loss curves of 100 test episodes whose values are divided by the initial loss of each episode. The shaded region represents the range $\pm$ standard deviation. HyperFlow generalizes well to three unseen domains by consistently moving the parameters to the regions with lower loss values.

Table 14: Ablation study on the diversity and construction of trajectories used to train HyperFlow.

Diversity Factor		Meta-Dataset Average		CD-FSL Average		
		OOD	ID	5w5s	5w20s	5w50s
# of Domains	1	59.71 ± 1.02	83.34 ± 0.62	61.80 ± 0.49	67.21 ± 0.41	68.83 ± 0.41
	4	60.27 ± 1.02	83.34 ± 0.62	62.44 ± 0.48	67.55 ± 0.42	69.53 ± 0.40
	8	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39
# of Episodes	40	60.64 ± 0.99	83.32 ± 0.63	62.12 ± 0.49	67.39 ± 0.42	68.34 ± 0.42
	4000	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39
# of Initializations	1	61.76 ± 0.98	83.33 ± 0.63	62.67 ± 0.48	67.33 ± 0.41	69.73 ± 0.39
	10	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39
# of Fine-Tuning steps	10	58.57 ± 0.97	83.80 ± 0.63	59.97 ± 0.50	65.61 ± 0.44	67.53 ± 0.42
	50	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39
Fine-Tuning Optimizer	RMSProp	58.10 ± 0.98	83.50 ± 0.63	60.96 ± 0.48	66.18 ± 0.42	68.09 ± 0.41
	Adam	63.11 ± 0.95	83.33 ± 0.63	62.82 ± 0.48	67.24 ± 0.41	69.60 ± 0.39



(a) (b)

Figure 6: Hyper-parameter sensitivity analysis.

to generate trajectories. Overall, performance tends to improve with broader coverage of both the task space and the parameter space, suggesting that diverse trajectories help the drift network learn adaptation patterns that transfer to unseen domains. At the same time, even the small-budget setting with 40 episodes still improves over Direct Transfer and Head-Tuning, indicating that HyperFlow does not require an excessively large trajectory set to obtain useful adaptation behavior.

We further study whether HyperFlow is sensitive to the fine-tuning procedure used to generate trajectories. When the offline trajectories are shortened from 50 fine-tuning steps to 10 steps, the resulting model still improves over Direct Transfer but shows lower CD-FSL performance than the default 50-step trajectory setting. This suggests that longer trajectories provide more informative supervision about the adaptation path, especially for modeling later-stage parameter dynamics. We also replace Adam with RMSProp during trajectory collection. The RMSProp-trained variant remains competitive but is generally below the default Adam-based trajectory setting on OOD and CD-FSL averages, indicating that HyperFlow is not tied to a single optimizer while still benefiting from trajectory sources that better match the target fine-tuning dynamics. Together, these results suggest that trajectory diversity and trajectory quality both affect performance: diverse tasks and initializations improve transfer, while sufficiently long and well-behaved fine-tuning paths provide stronger supervision for learning the conditional drift field.

Table 15: Few-shot classification accuracy (%) on CD-FSL benchmark with Multiple Drift model.

Model Variant	Average			EuroSAT			ISIC			CropDisease			ChestX		
	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s	5w5s	5w20s	5w50s
Single Drift	62.82	67.24	69.60	86.92	91.25	91.91	46.16	53.31	57.79	93.05	95.30	96.23	25.14	29.09	32.47
Bi-Drift	62.81	67.39	68.75	86.06	90.26	91.49	46.84	55.56	57.06	92.42	95.29	95.84	25.94	28.44	30.59

Hyper-parameter Sensitivity. During adaptation, HyperFlow has two hyper-parameters in the ODE solver: the step size and the number of solve steps. These respectively correspond to the learning rate and the number of gradient steps in the fine-tuning baselines, and we select them using a small validation set from each test dataset, following the learning-rate selection protocol of Hu et al. [18]. To analyze sensitivity to these choices, we evaluate the performance of HyperFlow-C and the fine-tuning baselines on CD-FSL 5-shot tasks while varying either the step size (learning rate) or the number of steps. As shown in Figure 6, HyperFlow exhibits a stable performance region and consistently outperforms Direct Transfer and Head-Tuning across a broad range of step sizes and step counts, supporting that our gains do not rely on brittle hyper-parameter choices.

Multiple Drift Models on ViT-S QKV Bias. We further examine whether HyperFlow benefits from a more fine-grained drift parameterization. Specifically, we compare the default single-drift model, which predicts the updates for all selected QKV-bias parameters jointly, with a multi-drift variant that assigns separate drift networks to different layer groups. While the multi-drift design may provide greater flexibility for modeling layer-specific update patterns, Table 15 shows that it does not lead to a clear or consistent performance gain. The results are largely comparable to the single-drift baseline, with small variations across shot settings and datasets.

These results suggest that the main bottleneck is unlikely to be the use of a single drift decoder. Instead, jointly predicting the QKV-bias updates appears to provide a useful inductive bias by sharing statistical strength across layers and tasks. Since the multi-drift variant also introduces additional auxiliary parameters and training complexity without clear empirical benefit, we adopt the simpler single-drift design in our main experiments.

F Additional Results

HyperFlow without Meta-Training. Our main experiments follow the P>M>F pipeline [18], where the target backbone is episodically meta-trained before adaptation. We note that the offline training of the drift network is intrinsic to HyperFlow, whereas the episodic meta-training of the target backbone is not: HyperFlow only assumes a fixed initialization from which fine-tuning trajectories can be collected. To verify this, we remove the meta-training stage and apply the same HyperFlow procedure directly to the pretrained DINO [32] ViT-S and BEiTv2 [64] ViT-L checkpoints. Specifically, we collect fine-tuning trajectories and train the drift network starting from each pretrained-only initialization. As shown in Table 16, HyperFlow consistently improves over Direct Transfer on both backbones and benchmarks even without meta-training. This indicates that HyperFlow can also be applied to pipelines consisting of pretraining and fine-tuning only, which have been shown to be effective for few-shot image classification [67], by replacing the gradient-based fine-tuning with the learned test-time adaptation.

Table 16: Results of HyperFlow applied to pretrained backbones without the meta-training stage.

Pretrained Backbone	Benchmark	Direct Transfer	HyperFlow-C	Bias-Tuning	Full-Tuning
DINO ViT-S	Meta-Dataset OOD	54.88 ± 1.01	60.63 ± 0.99	75.49 ± 0.79	77.18 ± 0.81
	CD-FSL 5-shot	63.61 ± 0.47	64.37 ± 0.46	65.39 ± 0.46	66.23 ± 0.45
BEiTv2 ViT-L	Meta-Dataset OOD	33.73 ± 0.95	71.58 ± 0.85	81.74 ± 0.68	81.11 ± 0.72
	CD-FSL 5-shot	54.30 ± 0.62	65.23 ± 0.47	65.62 ± 0.50	64.93 ± 0.52

Comparison with CLIP. We compare HyperFlow with a frozen OpenAI CLIP [68] ViT-B/16 model on exactly the same CD-FSL 5-way 5-shot test episodes. For zero-shot CLIP, we ignore the

labeled support images and perform no target-domain adaptation. For each episode, we encode the query images with the CLIP image encoder and construct text embeddings of the five sampled classes with the CLIP text encoder, using fixed domain-specific prompt templates: "a centered satellite photo of {}" for EuroSAT, "a dermoscopic photo of {}" for ISIC, "a photo of a plant leaf with {}" for CropDisease, and "a chest x-ray showing {}" for ChestX. Each query is assigned to the class whose text embedding has the highest cosine similarity. We also report few-shot CLIP, which uses the same CLIP image encoder but classifies the queries with ProtoNet-style class prototypes computed from the support images.

As reported in Table 17, zero-shot CLIP remains below all the support-conditioned methods on all four domains, despite having access to semantic class names that are not used by our prototype-based models. Few-shot CLIP performs substantially better than zero-shot CLIP, indicating that the low performance of zero-shot CLIP stems from the zero-shot inference setting rather than from an inherent limitation of the pretrained backbone. These results suggest that generic vision-language priors alone do not replace adaptation with labeled support examples in this cross-domain few-shot setting.

Table 17: Comparison with zero-shot and few-shot CLIP on CD-FSL 5-way 5-shot tasks.

Model	Support Images	Test-time backprop.	Average	EuroSAT	ISIC	CropDisease	ChestX
Zero-Shot CLIP (ViT-B)	No	No	39.84 ± 0.53	64.70 ± 0.52	23.18 ± 0.30	50.62 ± 1.05	20.86 ± 0.24
Few-Shot CLIP (ViT-B)	Yes	No	54.38 ± 0.54	72.62 ± 0.58	38.48 ± 0.54	84.19 ± 0.67	22.24 ± 0.38
Direct Transfer (ViT-S)	Yes	No	56.53 ± 0.54	80.56 ± 0.60	34.89 ± 0.51	85.80 ± 0.60	24.88 ± 0.44
HyperFlow-C (ViT-S)	Yes	No	62.82 ± 0.48	86.92 ± 0.44	46.16 ± 0.62	93.05 ± 0.42	25.14 ± 0.43

G Comprehensive Summary of the Computation Cost

In Tables 18–20, we provide the comprehensive analysis on the computation cost of the baselines in each of the three backbones, including FLOPs, peak memory, and the number of parameters. We measure each metric in a single NVIDIA A6000 GPU.

Table 18: Comprehensive analysis of the computation cost required by the adaptation methods on ViT-S backbone.

Model	Euler/Adam steps	FLOPs (G)	Peak Memory (MB)	Parameters (M)
Direct Transfer	-	74.89	361.62	21.67
HyperFlow	1	104.59	694.79	45.00
	10	104.81		
	50	105.76		
Head-Tuning	1	146.9	510.04	21.68
	2	218.91		
	5	434.93		
	10	794.98		
	20	1,515.06		
	50	3,675.32		
Bias-Tuning	1	218.91	5,039.52	21.67
	2	362.92		
	5	794.98		
	10	1,515.06		
	20	2,955.24		
	50	7,275.76		
LoRA-Tuning	1	230.56	7,916.21	22.85
	2	382.25		
	5	837.30		
	10	1,595.73		
	20	3,112.58		
	50	7,663.13		
Full-Tuning	1	218.91	8,038.23	21.67
	2	362.92		
	5	794.98		
	10	1,515.06		
	20	2,955.24		
	50	7,275.76		

Table 19: Comprehensive analysis of the computation cost on ViT-L backbone.

Model	Euler/Adam steps	FLOPs (G)	Peak Memory (MB)	Parameters (M)
Direct Transfer	-	1,034.73	1,765.11	303.21
HyperFlow	1	1,064.48	3,125.52	353.71
	10	1,065.18		
	50	1,068.31		
Head-Tuning	1	2,029.65	2,970.66	303.22
	2	3,024.58		
	5	6,009.37		
	10	10,984.01		
	20	20,933.29		
	50	50,781.14		
Bias-Tuning	1	3,024.58	27,307.70	303.21
	2	5,014.44		
	5	10,984.01		
	10	20,933.29		
	20	40,831.85		
	50	100,527.55		
LoRA-Tuning	1	3,071.20	39,385.78	309.50
	2	5,091.73		
	5	11,153.31		
	10	21,255.95		
	20	41,461.22		
	50	102,077.04		
Full-Tuning	1	3,024.58	49,156.63	303.21
	2	5,014.44		
	5	10,984.01		
	10	20,933.29		
	20	40,831.85		
	50	100,527.55		

Table 20: Comprehensive analysis of the computation cost on ConvNeXt-S backbone.

Model	Euler/Adam steps	FLOPs (G)	Peak Memory (MB)	Parameters (M)
Direct Transfer	-	147.77	1,026.04	49.45
HyperFlow	1	177.45	1,319.35	70.13
	10	177.62		
	50	178.36		
Head-Tuning	1	289.86	1,224.62	49.48
	2	431.94		
	5	858.21		
	10	1,568.64		
	20	2,989.51		
	50	7,252.12		
Bias-Tuning	1	431.94	7,159.56	49.45
	2	716.12		
	5	1,568.64		
	10	2,989.51		
	20	5,831.25		
	50	14,356.47		
Full-Tuning	1	431.94	22,708.68	49.45
	2	716.12		
	5	1,568.64		
	10	2,989.51		
	20	5,831.25		
	50	14,356.47		

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: We mention our scope as efficient test-time adaptation in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss our limitation in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

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 does not include any theoretical result.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

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: We describe implementation details in Section 5.1, Appendix B, C, and D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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: The code is available at `https://github.com/GitGyun/HyperFlow`.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

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: We describe implementation details in Section 5.1, Appendix B, C, and D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

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: We report both mean and standard deviation of the results in tables.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

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: We provide FLOPs and peak memory, the number and type of GPU used in the experiments in Section 5.3 and Appendix E and G.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

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: We follow the code of ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We discuss practical implication of our method throughout the paper.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

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 paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

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 (Meta-Dataset and CD-FSL) and pretrained models (DINO, BEiTv2, ConvNeXt, and CLIP) used in this paper are publicly available for research purposes. We cite their original papers and use them in accordance with their licenses and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: We do not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

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: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.