Flow to Learn: Flow Matching on Neural Network Parameters

Daniel Saragih
University of Toronto
daniel.saragih@mail.utoronto.ca

Deyu Cao
University of Toronto, University of Tokyo
deyu.cao@mail.utoronto.ca

Tejas Balaji
University of Toronto
tejas.balaji@mail.utoronto.ca

Ashwin Santhosh
University of Toronto
ashwin.santhosh@mail.utoronto.ca

***Abstract*—Foundational language models show a remarkable ability to learn new concepts during inference via context data. However, similar work for images lag behind. To address this challenge, we introduce FLoWN, a flow matching model that learns to generate neural network parameters for different tasks. Our approach models the flow on latent space, while conditioning the process on context data. Experiments verify that FLoWN attains various desiderata for a meta-learning model. In addition, it matches or exceeds baselines on in-distribution tasks, provides better initializations for classifier training, and is performant on out-of-distribution few-shot tasks while having a fine-tuning mechanism to improve performance.**

I. Introduction

Flow matching (FM) [1]–[3] is a prominent fixture in generative modeling tasks from imaging [2], [4]–[6] to language [7]–[9]. However, its application to neural network weights remains largely unexplored. In this paper, we introduce Flow-based Learning of Weights for Neural adaptation (FLoWN), a new class of method for weight generation. Empirical evaluations validate the following contributions: **1)** The generated weights match or exceed conventionally trained models on in-distribution tasks, and provide better initializations for fine-tuning on out-of-distribtion (OOD) tasks, **2)** FLoWN is able to conditionally retrieve pre-trained weights from a distribution pre-trained on various datasets while matching their performance, **3)** FLoWN is capable of performing well on OOD few-shot tasks while having a fine-tuning mechanism to improve performance.

A. Motivation

Multiple approaches have been tried to generate weights capable of few-shot learning (FSL), motivated by its speed compared to conventional training. For instance, various diffusion-based approaches [10]–[12] have been used to generate neural network weights. However, flexibility is limited by its restriction to Gaussian processes and a sluggish inference speed.

More broadly, we may categorize this form of learning as meta-learning [13]–[15], which aims to learn concepts from a few demonstrations. It is therefore natural that we have two evaluation settings: in-distribution tasks and out-of-distribution (OOD) tasks. With enough training and capacity,

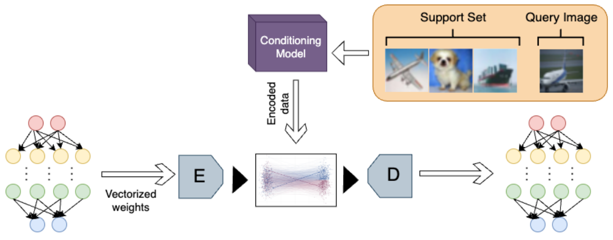

Fig. 1. A schematic of the training process of FLoWN for few-shot learning. Given a set of pre-trained target weights and a support set, we apply the conditioned flow model to pushforward a sample of the latent prior towards encoded target weights. The decoder is used during inference where we start from a sample of the latent prior and pushforward towards the target distribution with a trained vector field $v_\theta(\cdot, t; \boldsymbol{y})$ where $\boldsymbol{y}$ is the support set embedding.

it's clear *meta-models* (i.e. models trained on multiple data distributions) should excel at in-distribution tasks. However, generalization to novel tasks often presents a challenge to meta-learning and weight generation frameworks [12], [16]. Addressing this gap – improving a model's capacity to adapt beyond it's training distribution – serves as a key motivation for our work.

B. Related Works

Numerous strategies have emerged to narrow the out-of-distribution generalization gap within the realm of meta-learning and weight generation. One promising approach, proposed by Soro et al. [10], leverages diffusion-based models to enhance weight generation. While this method makes strides in bridging the generalization gap, we find that further improvements can be made, particularly by exploring alternative generative frameworks. Our proposed FLoWN framework builds on recent advances in conditional flow modeling, neural network parameter generation, and meta-learning to provide a more efficient and principled solution. Below, we review key developments in these areas; for additional context and related methods, see Appendix A.

a) Conditional Flow Matching: Lipman et al. [2] introduced the CFM objective, which learns probability paths between distributions using a conditional vector field. By modifying the coupling of the source and target distributions, later work shows better alignment with optimal transport paths, improving inference efficiency [4]. However, the original formulations of flow matching assumed that the initial distributions were Gaussian. Pooladian et al. [17] extended the theory to arbitrary source distributions using minibatch sampling and proved a bound on the variance of the gradient of the objective.

b) Neural Network Parameter Generation: Denil et al. [18] showed that most neural network parameters are redundant, enabling weight generation techniques. Ha et al. [19] introduced Hypernetworks, which generate weights using neural network layer embeddings. More recently, Wang et al. [12] and Soro et al. [10] applied latent diffusion models to parameter generation.

c) Weight Generation for Few-Shot Learning: Few-shot learning applies meta-learning to scenarios with limited data. Ravi & Larochelle [20] introduced an LSTM-based meta-learner for dynamic weight updates. Later works [13]–[15], [21] leveraged transformers and foundation models. Diffusion models for weight generation [10]–[12], [22] have gained attention, though primarily for in-distribution tasks, leaving OOD adaptation an open challenge.

C. Problem Definition

The problem of interest is that of *conditional weight generation* with an application to few-shot learning. Our approach trains a *conditional flow model* that can generate neural network weights tailored to new tasks from minimal data. At a high level, continuous flows [1]–[3], [23] provide a way to transform one distribution into another by modeling the dynamics of an ordinary differential equation (ODE). For our purposes: **1)** We define a reference flow that connects a simple "source" distribution to the "target" distribution of interest (in this case, the space of task-specific weights). **2)** A learnable velocity field (parameterized by a neural network v_θ) is trained to match the velocity of this reference flow, effectively bridging the source and target distributions. **3)** *Context-conditioning* is used to incorporate information about the task, such as the support set in an *n*-way-*k*-shot FSL setting (detailed below), so that the generated weights are adapted to the given task.

In few-shot classification, the standard setup involves *n* classes, each with *k* labeled examples (the "support set"). The learning system must then generalize to a set of unlabeled query examples from the same or new distribution. By conditioning the flow model on the support set (and any associated label embeddings), we aim to generate a set of weights that perform well immediately, while also allowing further fine-tuning when facing OOD data.

II. Methodology

A. Preliminaries

a) Conditional flow models: Chen et al. [23] first introduced continuous normalizing flows as an effective data generation process through modeling dynamics. Simulation-free methods improve on this concept by simplifying the training objective [1]–[3]. Following the formulation of Lipman et al. [2], given random variables $\bar{\mathbf{x}}_0 \sim p_0$ and $\bar{\mathbf{x}}_1 \sim p_1$ a data distribution, define a reference flow $\bar{\mathbf{x}} = (\bar{\mathbf{x}}_t)_{t\in[0,1]}$ where $\bar{\mathbf{x}}_t = \beta_t \bar{\mathbf{x}}_0 + \alpha_t \bar{\mathbf{x}}_1$ with the constraint that $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$. The aim of flow modeling is to learn a sequence $\mathbf{x} = (\mathbf{x}_t)_{t\in[0,1]}$ which has the same marginal distribution as $\bar{\mathbf{x}}$. To make this a feasible task, we describe this process as an ODE: $d\mathbf{x}_t = v(\mathbf{x}_t, t)dt$ where $\mathbf{x}_0 \sim \mathcal{N}(0, \boldsymbol{I})$. Training proceeds by first parameterizing $v(\mathbf{x}_t, t)$ by a neural network θ and matching the reference flow velocity, i.e. $u(\mathbf{x}_t, t) := \frac{d}{dt}\bar{\mathbf{x}}_t$. This would, however, be an unfeasible training objective, therefore, we condition on samples from the terminal distribution $\mathbf{x}_1 \sim p_1$ and train

$$L_{\text{cfm}}(\theta) = \mathbb{E}_{\text{t}\sim U[0,1], \mathbf{x}_1 \sim p_1, \mathbf{x}_t \sim p_t(\cdot|\mathbf{x}_1)} ||v_\theta(\mathbf{x}_t, t) - u(\mathbf{x}_t, t, \mathbf{x}_1)||. \tag{1}$$

Lipman et al. [2] proved that this loss produces the same gradients as the marginal loss, thus optimizing it will result in convergence to the reference $u(\mathbf{x}_t, t)$. Moreover, we can always marginalize an independent conditioning variable $\boldsymbol{y}$ on v_θ, u – this will serve as our context conditioning vector.

b) Few-shot learning: The problem of few-shot learning is often formulated as a n-way-k-shot classification task. In particular, given n classes and k examples for each class, $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{nk}$ of (image, label) *support* set pairs, our meta-learner is tasked with classifying *query* set images $\mathcal{Q} = \{\boldsymbol{x}_{nk+1}, \ldots, \boldsymbol{x}_{nk+q}\}$. Relevant to our approach, recently, Fifty et al. [13] proposed an image meta-learning architecture, CAML, consisting of three components: a frozen pre-trained image encoder, a class encoder, and a transformer-based sequence model. ELMES, the class encoder, was shown to possess two attractive properties: label symmetry and permutation invariance. The transformer sequence model takes the concatenation of the image and label embeddings of the support set images, and a special "unknown" label embedding is used on query set images. These query images are analagous to the [CLS] tokens in transformers as the logits corresponding to the query images are then passed into a classifier MLP to predict labels.

B. Flow to Learn

We describe the components of our approach below, alongside a method schematic in Figure 1, and leave more details to Appendix B, C.

a) Weight encoder: Due to the intractable size of weight space, it is necessary for modeling to take place in latent space. We justify this design by appealing to work on the Lottery Ticket Hypothesis [24], [25] as well as the body of work on pruning [26], which suggests that, like natural data, neural networks live on a low-dimensional manifold within its ambient space. We have two encoder variants, first is a variational autoencoder (VAE) [27] set up as in Soro et al. [10], and the second is the graph-based encoder (GE) of Kofinas et al. [28] which takes into account permutation invariance of neural networks. As the latter models connections between layers, we only use the VAE for experiments involving subsets of weights.

TABLE I
MEAN VALIDATION ACCURACY OF UNCONDITIONAL FLOWN GENERATION. *orig* DENOTES BASE MODELS TRAINED CONVENTIONALLY AND *p-diff* THOSE GENERATED USING P-DIFF [12].

	CIFAR100			CIFAR10			MNIST			STL10		
Base Models	orig.	FLoWN	p-diff.	orig.	FLoWN	p-diff.	orig.	FLoWN	p-diff	orig.	FLoWN	p-diff
Resnet-18	71.45	71.42	71.40	94.54	94.36	94.36	99.68	99.65	99.65	62.00	62.00	62.24
ViT-base	85.95	85.86	85.85	98.20	98.11	98.12	99.41	99.38	99.36	96.15	95.77	95.80
ConvNext-tiny	85.06	85.12	85.17	98.03	97.89	97.90	99.42	99.41	99.40	95.95	95.63	95.63
CNN w/ GE	32.09	31.73	31.81	72.53	72.15	72.09	98.93	98.89	98.89	53.88	53.64	53.80

TABLE II
MEAN VALIDATION ACCURACY OF TOP-5 FLOWN MODEL RETRIEVALS.

Method	MNIST	F-MNIST	CIFAR10	STL10
Original	91.1	72.7	48.7	39.0
FLoWN w/ mIN-prior	63.0	41.9	22.6	18.8
FLoWN	91.7	73.8	50.3	40.8

b) Flow meta-model: The backbone of our meta-learning framework is a conditional FM model following Tong et al. [4]. We make use of the flexibility of FM to use a non-Gaussian prior, specifically the Kaiming uniform or normal initializations [29], as the source p_0. The data distribution p_1 of base model weights is experiment-dependent, however, broadly they are obtained by conventional training methods or through a model zoo [30].

c) Conditioning model: To condition our flow meta-model, we use a pre-trained CAML. Our choice is due to the extensive training of the CAML architecture on several datasets as well as the principled label encoding used in their approach. As we are only interested in encoding the support set, we consider judicious choices for the query set expected by CAML. For instance, in the Model Retrieval experiment below, we simply choose one random image for each class in the support set. In the FSL experiment, the choice is clear: the query set of each FSL task. The conditioning vector is incorporated by concatenating to the latent vector.

III. RESULTS

First, we confirm various properties that are to be expected of weight generation models. Next, we examine FLoWN's performance in few-shot learning. Further details are provided in Appendix D.

A. Basic Properties of FLoWN

a) Unconditional generation: We first evaluate the basic modeling capabilities of the flow meta-model. The target distribution p_1 is generated by training a variety of base models on known datasets: CIFAR-10, CIFAR-100, and MNIST, and saving 200 weight checkpoints each. For large models, we can choose to generate only a subset of the weights. In our case, we generate the batch norm parameters for Resnet-18 [31], ViT-base [32] and ConvNext-tiny [33], and the full medium-CNN [30]. The aim of this test is to train a separate meta-model for each dataset and validate its base model reconstruction on classifying its corresponding test set. Table I shows that we are able to match base models trained conventionally and with p-diff [12].

b) Model retrieval and in-distribution initialization: Following [10], we perform model retrieval to test whether the meta-model can distinguish weights of the base model given conditioning samples from the dataset the base model was trained on. The base model is a simple 4-layer ConvNet and we obtain 100 weight checkpoints from the model zoo [30] for each dataset: MNIST, Fashion-MNIST (F-MNIST), CIFAR-10, and STL10 after 46-50 epochs of conventional training. Unlike in the previous test, we will train just a single meta-model on 400 total base models conditioned on support samples from their training set via CAML. During validation, we pass in a random support sample from one of the four datasets and generate the *full* ConvNet. In Table II, we see that our top-5 validation accuracy matches that of the base models. Additionally, we repeated this experiment using weights from mini-Imagenet as a prior, but they seem to perform considerably worse than just Kaiming normal (see Appendix D2 for a discussion). Next, we repeat this experiment but instead using weight checkpoints from epochs 21-25, and use the generated weights as an initialization before fine-tuning another 25 epochs. As shown in Table III, our initialization enjoys faster convergence, even for datasets on which the model was not trained, highlighting the generalization capability of our meta-model.

c) Fine-tuning the meta-model on OOD data: The setting of unconditional generation is quite restrictive as it is assumed that the output classifier has the same architecture and is to be used on the same dataset. In this experiment, we evaluate whether the meta-model can be effectively fine-tuned to achieve better performance on out-of-distribution data. We start with the meta-model trained from unconditional generation and generate weights for a different dataset. Subsequently, we compute the cross-entropy loss and backpropagate the gradients through the FM model. As this entails backpropagation through an ODE solver, we implement a stopgrad mechanism that restricts gradient flow before a time $0 < t' < 1$ to trade off accuracy for efficiency. Due to time constraints, we restrict ourselves to batch norms of Resnet-18 and the small-CNN. Table IV shows considerable improvement over generations obtained from a static FM meta-model and the VAE.

TABLE III
Mean validation accuracy of fine-tuned generated weights post-retrieval. The asterisk (*) indicates datasets on which the model was not trained.

Epoch	Method	MNIST	F-MNIST	CIFAR10	STL10	USPS*	SVHN*	KMNIST*
0	RandomInit	$\sim 10\%$	$\sim 10\%$	$\sim 10\%$	$\sim 10\%$	$\sim 10\%$	$\sim 10\%$	$\sim 10\%$
	FLoWN	83.58 ± 0.58	68.50 ± 0.64	45.93 ± 0.57	35.16 ± 1.24	57.53 ± 2.43	17.99 ± 0.82	11.79 ± 0.51
1	RandomInit	18.12 ± 1.58	26.90 ± 0.52	28.75 ± 0.22	18.94 ± 0.09	17.69 ± 0.00	19.50 ± 0.03	14.48 ± 0.06
	FLoWN	84.49 ± 0.65	69.09 ± 0.40	46.85 ± 0.30	36.15 ± 1.14	72.45 ± 1.81	68.64 ± 7.07	51.15 ± 8.90
5	RandomInit	35.05 ± 3.87	51.08 ± 2.15	40.00 ± 0.20	28.24 ± 0.01	32.77 ± 0.46	39.59 ± 10.0	30.00 ± 0.30
	FLoWN	87.68 ± 0.44	70.32 ± 0.50	47.44 ± 0.55	37.43 ± 1.19	76.96 ± 1.29	77.36 ± 1.07	69.14 ± 10.1
25	RandomInit	87.70 ± 0.90	70.69 ± 0.46	46.86 ± 0.01	36.75 ± 0.10	82.02 ± 0.12	58.56 ± 19.5	55.05 ± 0.06
	FLoWN	92.29 ± 0.41	73.72 ± 0.68	49.25 ± 0.73	40.14 ± 1.07	82.28 ± 1.40	78.75 ± 1.30	79.11 ± 6.65
50	RandomInit	92.76 ± 0.08	72.88 ± 0.46	48.85 ± 0.74	40.47 ± 0.18	88.35 ± 0.18	63.70 ± 22.1	64.32 ± 0.25

TABLE IV
Fine-tuning on OOD data. Data-F are results generated from fine-tuned meta-models, whereas Data-S are from static meta-models. Here, we generate the full CNN weights, whereas we only modify the batch norms of ResNet-18.

	ResNet-18		CNN w/ VAE		CNN w/ GE	
Base dataset	CIFAR10	STL10	CIFAR10	STL10	CIFAR10	STL10
CIFAR10-S	–	64.20	–	24.01	–	23.03
CIFAR10-F	–	72.87	–	60.09	–	60.85
STL10-S	93.09	–	19.97	–	18.13	–
STL10-F	94.06	–	61.38	–	69.42	–

TABLE V
Few-shot learning accuracy on out-of-distribution tasks. We compare with D2NWG and best weights generated by FLoWN.

Model	CIFAR-10	STL10
FLoWN-best	**73.1**	**80.4**
D2NWG [10]	33.04 ± 0.04	50.42 ± 0.13
FLoWN	35.84 ± 2.71	35.35 ± 2.77

B. Few-shot learning

For this evaluation, we utilize mini-Imagenet [34] and Chen et al. [35] for a Resnet-12 architecture. Following the typical FSL setting, we partition the dataset into meta-train and meta-test sets and further into tasks whose size depends on the way and shot parameters. For instance, for 5-way-1-shot, the support set consists of one image from 5 different classes whereas the query set is always 15 images for each of the 5 classes in the support set. The in-distribution test entails labeling query images from the same dataset (i.e. trained on mini-Imagenet and evaluated on mini-Imagenet), whereas OOD tasks entails labeling novel query images. First, we train the Resnet-12 on the train split of mini-Imagenet; our goal is thus to generate a classifier head for each task.

In our case, we set 50,000 tasks in the meta-train set and 100 in the meta-test set. We perform this test by constructing a target distribution using pre-trained weights from Resnet-12, linear-probing a classifier head on top of the Resnet backbone for each of the 50,000 subsets for 100 epochs using the AdamW optimizer with a learning rate of 10^{-3} and weight decay of 10^{-2}. Given our computational constraints, we evaluated FLoWN on just two out-of-distribution datasets: CIFAR10 and STL10 by sampling weights 50 different times and taking the average top-3 accuracies. Table V shows that our method achieves marginal gains on CIFAR10, but performance on STL10 remains below the baseline. Considering the high validation accuracies of our VAE, we anticipate that further training and tinkering will enhance FLoWN generalization across tasks.

IV. Conclusion

In this work, we have provided a preliminary investigation of FLoWN for weight generation with an application to few-shot learning. Future research directions include: **1)** training FLoWN on a more comprehensive image dataset to improve efficacy on OOD tasks, **2)** a post-hoc fine-tuning mechanism [36] for adapting FLoWN to difficult domains (e.g. medical imaging), **3)** incorporating intermediate base model weights obtained during conventional training to guide the inference trajectory of generated weights (e.g. via MetricFM [37]).

References

[1] M. S. Albergo and E. Vanden-Eijnden, "Building normalizing flows with stochastic interpolants," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=li7qeBbCR1t

[2] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow Matching for Generative Modeling," Feb. 2023.

[3] X. Liu, C. Gong, and qiang liu, "Flow straight and fast: Learning to generate and transfer data with rectified flow," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=XVjTT1nw5z

[4] A. Tong, K. Fatras, N. Malkin, G. Huguet, Y. Zhang, J. Rector-Brooks, G. Wolf, and Y. Bengio, "Improving and generalizing flow-based generative models with minibatch optimal transport," Mar. 2024.

[5] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, K. Lacey, A. Goodwin, Y. Marek, and R. Rombach, "Scaling rectified flow transformers for high-resolution image synthesis," 2024. [Online]. Available: https://arxiv.org/abs/2403.03206

[6] X. Liu, X. Zhang, J. Ma, J. Peng, and qiang liu, "Instaflow: One step is enough for high-quality diffusion-based text-to-image generation," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=1k4yZbbDqX

[7] I. Gat, T. Remez, N. Shaul, F. Kreuk, R. T. Q. Chen, G. Synnaeve, Y. Adi, and Y. Lipman, "Discrete flow matching," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: https://openreview.net/forum?id=GTDKo3Sv9p

[8] N. Shaul, I. Gat, M. Havasi, D. Severo, A. Sriram, P. Holderrieth, B. Karrer, Y. Lipman, and R. T. Q. Chen, "Flow matching with general discrete paths: A kinetic-optimal perspective," 2024. [Online]. Available: https://arxiv.org/abs/2412.03487

[9] A. Campbell, J. Yim, R. Barzilay, T. Rainforth, and T. Jaakkola, "Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design," *arXiv preprint arXiv:2402.04997*, 2024.

[10] B. Soro, B. Andreis, H. Lee, S. Chong, F. Hutter, and S. J. Hwang, "Diffusion-based Neural Network Weights Generation," Feb. 2024.

[11] B. Zhang, C. Luo, D. Yu, X. Li, H. Lin, Y. Ye, and B. Zhang, "Metadiff: Meta-learning with conditional diffusion for few-shot learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 15, pp. 16 687–16 695, Mar. 2024.

[12] K. Wang, Z. Xu, Y. Zhou, Z. Zang, T. Darrell, Z. Liu, and Y. You, "Neural Network Diffusion," Feb. 2024.

[13] C. Fifty, D. Duan, R. G. Junkins, E. Amid, J. Leskovec, C. Re, and S. Thrun, "Context-Aware Meta-Learning," Mar. 2024.

[14] S. X. Hu, D. Li, J. Stühmer, M. Kim, and T. M. Hospedales, "Pushing the Limits of Simple Pipelines for Few-Shot Learning: External Data and Fine-Tuning Make a Difference," Apr. 2022.

[15] A. Zhmoginov, M. Sandler, and M. Vladymyrov, "HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning," Jul. 2022.

[16] K. Schürholt, M. W. Mahoney, and D. Borth, "Towards scalable and versatile weight space learning," in *Proceedings of the 41st International Conference on Machine Learning (ICML)*. PMLR, 2024.

[17] A.-A. Pooladian, H. Ben-Hamu, C. Domingo-Enrich, B. Amos, Y. Lipman, and R. T. Q. Chen, "Multisample flow matching: straightening flows with minibatch couplings," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023.

[18] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting Parameters in Deep Learning," Oct. 2014.

[19] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *International Conference on Learning Representations*, 2017.

[20] S. Ravi and H. Larochelle, "Optimization as a Model for Few-Shot Learning," in *International Conference on Learning Representations*, Feb. 2017.

[21] L. Kirsch, J. Harrison, J. Sohl-Dickstein, and L. Metz, "General-purpose in-context learning by meta-learning transformers," 2024. [Online]. Available: https://arxiv.org/abs/2212.04458

[22] Y. Du, Z. Xiao, S. Liao, and C. Snoek, "ProtoDiff: Learning to Learn Prototypical Networks by Task-Guided Diffusion," Nov. 2023.

[23] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural Ordinary Differential Equations," Dec. 2019.

[24] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[25] B. Liu, Z. Zhang, P. He, Z. Wang, Y. Xiao, R. Ye, Y. Zhou, W.-S. Ku, and B. Hui, "A survey of lottery ticket hypothesis," 2024. [Online]. Available: https://arxiv.org/abs/2403.04861

[26] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10 558–10 578, 2024.

[27] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022. [Online]. Available: https://arxiv.org/abs/1312.6114

[28] M. Kofinas, B. Knyazev, Y. Zhang, Y. Chen, G. J. Burghouts, E. Gavves, C. G. M. Snoek, and D. W. Zhang, "Graph Neural Networks for Learning Equivariant Representations of Neural Networks," https://arxiv.org/abs/2403.12143v3, Mar. 2024.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[30] K. Schürholt, D. Taskiran, B. Knyazev, X. Giró-i Nieto, and D. Borth, "Model zoos: A dataset of diverse populations of neural network models," in *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, Sep. 2022.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:206594692

[32] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[33] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," 2022. [Online]. Available: https://arxiv.org/abs/2201.03545

[34] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf

[35] Y. Chen, Z. Liu, H. Xu, T. Darrell, and X. Wang, "Meta-baseline: Exploring simple meta-learning for few-shot learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9062–9071.

[36] C. Domingo-Enrich, M. Drozdzal, B. Karrer, and R. T. Q. Chen, "Adjoint Matching: Fine-tuning Flow and Diffusion Generative Models with Memoryless Stochastic Optimal Control," Sep. 2024.

[37] K. Kapusniak, P. Potaptchik, T. Reu, L. Zhang, A. Tong, M. Bronstein, A. J. Bose, and F. Di Giovanni, "Metric Flow Matching for Smooth Interpolations on the Data Manifold," May 2024.

[38] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 1126–1135.

[39] A. Antoniou, H. Edwards, and A. Storkey, "How to train your MAML," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HJGven05Y7

[40] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, *Meta-learning with implicit gradients*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[41] D. Zhao, S. Kobayashi, J. Sacramento, and J. von Oswald, "Meta-Learning via Hypernetworks," in *4th Workshop on Meta-Learning at NeurIPS 2020 (MetaLearn 2020)*. NeurIPS, Dec. 2020.

[42] M. Przewiezlikowski, P. Przybysz, J. Tabor, M. Zieba, and P. Spurek, "Hypermaml: Few-shot adaptation of deep models with hypernetworks," *ArXiv*, vol. abs/2205.15745, 2022.

[43] J. Beck, M. T. Jackson, R. Vuorio, and S. Whiteson, "Hypernetworks in Meta-Reinforcement Learning," in *Proceedings of The 6th Conference on Robot Learning*. PMLR, Mar. 2023, pp. 1478–1487.

[44] J. Lee, A. Xie, A. Pacchiano, Y. Chandak, C. Finn, O. Nachum, and E. Brunskill, "Supervised Pretraining Can Learn In-Context Reinforcement Learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 43 057–43 083, Dec. 2023.

[45] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," 2020. [Online]. Available: https://arxiv.org/abs/2004.05718

[46] C. Diao and R. Loynd, "Relational attention: Generalizing transformers for graph-structured tasks," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=cFuMmbWiN6

[47] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis*, 2009.

[48] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *AISTATS*, 2011.

[49] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[50] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. (2018) Deep learning for classical japanese literature.

[51] J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

[52] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[53] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

Appendix

This appendix consist of details left out in the main text. First, we perform a more comprehensive review of the related literature with further discussion of our motivations. Next, we go over the various components of FLoWN and expound on their implementation and training procedure.

A. Related Works

a) Conditional flow matching: The CFM objective, where a conditional vector field is regressed to learn probability paths from a source to target distribution, was first introduced in Lipman et al. [2]. The CFM objective attempts to minimize the expected squared loss of a target conditional vector field (which is conditioned on training data and generates a desired probability path) and an unconditional neural network. The authors showed that optimizing the CFM objective is equivalent to optimizing the unconditional FM objective. Moreover, the further work [4] highlighted that certain choices of parameters for the probability paths led to the optimal conditional flow being equivalent to the optimal transport path between the initial and target data distributions, thus resulting in shorter inference times. However, the original formulations of flow matching assumed that the initial distributions were Gaussian. Pooladian et al. [17] extended the theory to arbitrary source distributions using minibatch sampling and proved a bound on the variance of the gradient of the objective. Tong et al. [4] showed that using the 2-Wasserstein optimal transport map as the joint probability distribution of the initial and target data along with straight conditional probability paths results in a marginal vector field that solves the dynamical optimal transport problem between the initial and target distributions.

b) Neural network parameter generation: Due to the flexibility of neural network as function approximators, it is natural to think that they could be applied to neural network weights. Denil et al. [18] paved the way for this exploration as their work provided evidence of the redundancy of most network parameterizations, hence showing that paramter generation is a feasible objective. Later, Ha et al. [19] introduced Hypernetworks which use embeddings of weights of neural network layers to generate new weights and apply their approach to dynamic weight generation of RNNs and LSTMs. A significant portion of our paper's unconditional parameter generation section builds upon the ideas from Wang et al. [12] and the concurrent work of Soro et al. [10] where the authors employ a latent diffusion model to generate new parameters for trained image classification networks.

c) Meta-learning context: Although neural networks are adept at tasks on which they were trained, a common struggle of networks is generalization to unseen tasks. In contrast, humans can often learn new tasks when given only a few examples. A pioneering modern work in this field is MAML [38], which learns good initialization parameters for the meta-learner such that it can easily be fine-tuned to new tasks. Their approach utilizes two nested training loops. The inner loop computes separate parameters adapted to each of the training tasks. The outer loop computes the loss using each of these parameters on their respective tasks and updates the model's parameters through gradient descent. However, MAML often had unstable training runs, and so successive works gradually refined the method [39]–[42]. The aforementioned works typically focus on classification tasks, however, this paradigm allows for great versatility. For instance, Beck et al. [43] used hypernetworks to generate the parameters of a policy model and Lee et al. [44] exploited the in-context learning ability of transformers to general reinforcement learning tasks.

d) Weight generation for few-shot learning: Following up on the work of meta-learning context, few-shot learning is a natural application of such meta-learning algorithms. An early example is Ravi & Larochelle [20] who designed a meta-learner based on the computations in an LSTM cell. At each training example in the support set, their meta learner uses the losses and the gradients of the losses of the base learner (in addition to other information from previous training examples) to produce base learner parameters for the next training example. The loss of the base learner on the test examples in the support set is backpropagated through the meta learner's parameters. Moreover, we may leverage the advancements in generative modeling for weight generation. As we mentioned, Lee et al. [44] used transformers for in-context reinforcement learning, but we also see the works of Zhmoginov et al. [15]; Hu et al. [14]; Kirsch et al. [21]; Fifty et al. [13] use transformers and foundation models. More similar to our method is the body of work on using diffusion models for weight generation [10]–[12], [22]. These methods vary in their approach, some leveraging a relationship between the gradient descent algorithm and the denoising step in diffusion models to design their meta-learning algorithm. Others rely on the modeling capabilities of conditioned latent diffusion models to learn the target distribution of weights. Most evaluations conducted were in-distribution tasks, i.e. tasks sampled from the same data distribution as the training tasks, hence, there is room to explore ways of adapting this approach for out-of-distribution tasks.

B. Architecture Details

Here, we expound on the architecture of FLoWN. See Figure 1 for a schematic of the training and inference process.

1) Variational Autoencoder: The variational autoencoder follows the implementation of Soro et al. [10]. In particular, given a set of model weights $\{\mathcal{M}_i\}_{i=1}^N$, we first flatten the weights to obtain vectors $\boldsymbol{w}_i \in \mathbb{R}^{d_i}$. For the sake of uniformity, we always zero-pad vectors to $d = \max_i d_i$. Alternatively, we allow for layer-wise vectorization: set a chunk size ℓ which corresponds to the weight dimension of a network layer. Then, zero-pad $\boldsymbol{w}_i$ to be a multiple of ℓ, say $\tilde{d}$. This allows us to partition into k equal length vectors $\boldsymbol{w}_{i,k} \in \mathbb{R}^{\tilde{d}/k}$. Typically, larger models benefit from layer-wise vectorization.

Subsequently, we train a VAE to obtain an embedding of such vectors by optimizing the objective:

$$L_{\text{VAE}}(\theta, \phi) := -\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{w})}[\log p_\theta(\boldsymbol{w}|\boldsymbol{z}) + \beta D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{w})||p(\boldsymbol{z}))] \tag{2}$$

where $\boldsymbol{w}$ is the vectorized weights, $\boldsymbol{z}$ is the embedding we are learning, and p_θ, q_ϕ are the reconstruction and posterior distributions respectively. Moreover, we fix the prior $p(\boldsymbol{z})$ to be a Gaussian and the weight is set to be $\beta = 10^{-2}$. For layer-wise vectorization, we simply change the input dimensions to match the chunk size. Upon decoding, we concatenate the chunks to re-form the weight vector.

2) Graph Encoder: Recently, Kofinas et al. [28] proposed a neural graph encoder which incorporates the permutation invariance present in network weights. The method has two components: a graph constructor and the embedding model. The neural network is first represented as a graph where nodes represent the neurons within each network layer and edges represent neuronal connections. Importantly, node features correspond to bias parameters and edge features correspond to weight parameters. Subsequently, this is fed into an embedding model, such as a GNN, specifically PNA [45], or a relational transformer [46]. For our use case, we outline weights-to-graphs conversion of MLPs, CNNs, and normalization layers. See Kofinas et al. [28] for more details.

a) MLPs to graphs: Let $\mathcal{G}(\boldsymbol{V}, \boldsymbol{E})$ be a graph and let the vertex set $\boldsymbol{V} \in \mathbb{R}^{n \times d_V}$ and the adjacency matrix $\boldsymbol{E} \in \mathbb{R}^{n \times n \times d_E}$. Intuitively, if we have n nodes in a graph, our vertex set is size n, and the adjacency matrix is $n \times n$. In our case, we also incorporate node and edge features, hence an extra dimension is added. Consider an L-layer MLP with weights $\{\boldsymbol{W}^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}\}_{\ell=1}^L$ and biases $\{\boldsymbol{b}^\ell \in \mathbb{R}^{d_\ell}\}_{\ell=1}^L$. Since we have a node for each neuron, we have $n = \sum_{\ell=0}^L d_\ell$, where d_0 is the input dimension. Now, let's use these to construct the vertex set $\boldsymbol{V}$. Since each neuron has a corresponding bias term (except the input), $\boldsymbol{V} = [\mathbf{0_{d_0}} \ \mathbf{b^1} \ \ldots \ \mathbf{b^L}]^\top$. As for the adjacency matrix, consider the first d_0 rows: as this corresponds to the input layer, it's only connected to the first layer, i.e. only columns d_0+1 to d_0+d_1 are possibly non-zero. And if we focus on row $i \in [d_0]$, what are its features? They must be $\boldsymbol{W}^1_{:,i}$. Hence,

$$(\boldsymbol{E}_{[0:d_0]\times[d_0+1, d_0+d_1]})^\top = \boldsymbol{W}^1,$$

and elsewhere in $\boldsymbol{E}_{[0:d_0]}$ is zero. In general,

$$(\boldsymbol{E}_{[d_{i-1}:d_i]\times[d_{i-1}+1, d_{i-1}+d_i]})^\top = \boldsymbol{W}^i,$$

and is zero everywhere else. In other words, the first off-diagonal blocks are precisely $\boldsymbol{W}^i$, and $\boldsymbol{E}$ is zero elsewhere. Finally, what are d_E and d_V? This turns out to be problem-dependent. Sometimes, it helps to add useful node features, but if the only thing we are concerned about embedding is weight information, then each entry of $\boldsymbol{W}^i$ and $\boldsymbol{b}^i$ is simply a scalar, so $d_E = d_V = 1$.

b) Normalization layers to graphs: Either BatchNorm or LayerNorm can be written as $\boldsymbol{y} = \boldsymbol{m} \odot \boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{b}, \boldsymbol{y} \in \mathbb{R}^d$. The trick is to recast this as a linear layer: we can always write $\boldsymbol{y} = \text{diag}(\boldsymbol{m})\boldsymbol{x} + \boldsymbol{b}$. Hence, we ought to have d nodes for $\boldsymbol{x}$ and another d nodes for $\boldsymbol{y}$ where the nodes for $\boldsymbol{y}$ have biases $\boldsymbol{b}$. The two layers are then connected by weight matrix $\text{diag}(\boldsymbol{m})$ which only connects x_i to y_i.

TABLE VI
MODEL ARCHITECTURES AND HYPERPARAMETERS. SQUARE BRACKETS $[\cdot]$ INDICATES AN INTERVAL OF VALUES. FOR INSTANCE, WE OFTEN TRAIN UNTIL A LOSS PLATEAU, HENCE THE VARYING NUMBER OF EPOCHS.

Parameters	Model Retrieval	Few-Shot Learning
Dataset Encoder (Frozen)		
Architecture	CAML	CAML
Latent Dimension	1024	1024
Weight Encoder		
Architecture	VAE	VAE
Latent Space Size	$4 \times 4 \times 4$	$4 \times 8 \times 8$
Upsampling/Downsampling Layers	5	4
Channel Multiplication (per Downsampling Layer)	(1, 1, 2, 2, 2)	(1,1,2,2)
ResNet Blocks (per Layer)	2	2
KL-Divergence Weight	0.01	1e-6
Optimizer	AdamW	AdamW
Learning Rate	1×10^{-3}	1×10^{-2}
Weight Decay	2×10^{-6}	2×10^{-6}
Batch Size	32	128
Training Epochs	3000	[100, 500]
Conditional Flow Matching Model		
Timestep and Dataset Embedding Size	128	128
Input Size	$4 \times 4 \times 4$	$4 \times 8 \times 8$
Optimizer	AdamW	AdamW
Learning Rate	1×10^{-3}	2×10^{-4}
Weight Decay	2×10^{-6}	2×10^{-6}
Batch Size	32	128
Training Epochs	[3000, 10000]	[100, 500]

c) CNNs to graphs: To simplify consider one convolutional layer between layers $\ell - 1$ and ℓ, namely $\boldsymbol{W} \in \mathbb{R}^{d_\ell \times d_{\ell-1} \times w_\ell \times h_\ell}$ and $\boldsymbol{b} \in \mathbb{R}^{d_\ell}$. Intuitively, $d_{\ell-1}$ is the number of input channels and d_ℓ the number of output channels. Due to the spatial dimension $w_\ell \times h_\ell$, we first flatten the last two layers. Now, we make use of the node and edge features: instead of scalar weights like in linear layers, our weights are vectors of size $w_\ell \times h_\ell$. However, the size may be different between layers, so we take $s = (\max_{\ell \in [L]} w_\ell, \max_{\ell \in [L]} h_\ell)$ and zero-pad our weight vectors as necessary before flattening. Hence, following the procedure in the MLP conversion, we form an adjacency matrix with vector features, i.e. $\boldsymbol{E} \in \mathbb{R}^{n \times n \times d_E}$ where $d_E = w_{\max} h_{\max}$.

3) Flow Model: The neural network used for flow matching is the UNet from D2NWG [10]. The specific hyperparameters used for the CFM model varies between experiments, so we leave this discussion to D.

C. Training Details

Here, we present further training and experimental details.

1) Pre-trained Model Acquisition:

a) Datasets and architectures: We conduct experiments on a wide range of datasets, including CIFAR-10/100 [47], STL-10 [48], (Fashion/K)-MNIST [49], [50], USPS [51], and SVHN [52]. To evaluate our meta-model's ability to generate new subsets of network parameters, we conduct experiments on ResNet-18 [31], ViT-Base [32], ConvNeXt-Tiny [33], the latter two are sourced from timm Wightman [53]. As we shall detail below, small CNN architectures from a model zoo [30] are also used for full-model generations.

TABLE VII
PREPROCESSING AND GRAPH ENCODER HYPERPARAMETERS.

Parameters	Values
Dataset Preprocessing	
Input Channels	3
Image shape	(32, 32)
$(w_{\max}, h_{\max})$	(7, 7)
Max. spatial res.	49
Max. # hidden layers	5
Flattening Method	Repeat Nodes
Normalize	False
Augmentation	False
Linear as Conv.	False
Relational Transformer	
Embed dim.	64
Num. layers	4
Num. heads	8
Num. probe features	0

b) Model pre-training: For better control over the target distribution p_1, in experiments involving ResNet-18, ViT-Base, and ConvNeXt-Tiny, we pre-train these base models from scratch on their respective datasets. We follow Wang et al. [12] and train the base models until their accuracy stabilizes. Further, we train the relevant subset (e.g. batch norm parameters for ResNet-18) for another 200 epochs, saving the weights at the end.

c) Model zoo: The model zoo used for meta-training in the model retrieval setting, as described in Sec. III-A0c, was sourced from [30]. As the base model, we employed their CNN-small architecture, which consists of three convolutional layers and contains either 2,464 or 2,864 parameters, depending on the number of input channels. For each dataset—MNIST, Fashion-MNIST, CIFAR-10, and STL10—100 sets of pre-trained weights were randomly selected from the model zoo using different seeds and fixed hyperparameters (referred to as "Seed" in their codebase). For the training of base models, we adopted the same hyperparameters as those used in [30] for all datasets, except KMNIST, which was not included in their model zoo. For KMNIST, we used the hyperparameters applied to MNIST, given the similarity between the two datasets.

2) Variational Autoencoder Training: The VAE was trained with the objective in equation 2. Moreover, following p-diff [12], we add Gaussian noise to the input and latent vector, i.e. given noise factors σ_{in} and σ_{lat} with encoder f_ϕ and decoder f_θ, we instead have

$$\boldsymbol{z} = f_\phi(\boldsymbol{w} + \xi_{in}),\ \hat{\boldsymbol{w}} = f_\theta(\boldsymbol{z} + \xi_{lat})$$

where $\xi_{in} \sim \mathcal{N}(0, \sigma_{in}^2 \boldsymbol{I}),\ \xi_{lat} \sim \mathcal{N}(0, \sigma_{lat}^2 \boldsymbol{I})$.

A new VAE is trained at every instantiation of the CFM model as architectures often differ in their input dimension for different experiments. However, they are trained with different objectives: the VAE is trained to minimize reconstruction loss. In all experiments, we fix $\sigma_{in} = 0.001$ and $\sigma_{lat} = 0.5$.

TABLE VIII
TASK TRAINING

Parameters	ResNet18	ViT & ConvNext	CNN
Optimizer	SGD	AdamW	AdamW
Initial Training LR	0.1	1×10^{-4}	3×10^{-3}
Training Scheduler	MultiStepLR	CosineAnnealingLR	CosineAnnealingLR
Layer Weights Saved	Last 2 BN layers	Last 2 BN layers	All layers
Initial Model Saving LR	1.6×10^{-4}	5×10^{-2}	1×10^{-3}
Model Saving Scheduler	None	CosineAnnealingLR	CosineAnnealingLR
Number of Models Saved	200	200	200
Num. of Weights per Model	2048	3072	[10565, 12743]
Training Epochs	100	100	100
Batch Size	64	128	128

3) Graph Encoder Training: The graph encoder [28] was used for both the unconditional generation and fine-tuning on OOD experiments with the CNN-medium architecture from Schürholt et al. [30]. We restricted our tests to the relational transformer [46] which was shown to perform better in the original paper [28]. See Table VII for the instantiation parameters.

D. Experimental Details

1) Unconditional Generation: Unconditional generation involves two stages: first is the training of base models. We choose a Resnet18, ViT-B, ConvNext-tiny, and medium-CNN for our base models and provide the training parameters in Table VIII. Next, is the stage where we train either a AE-CFM or AE-DDPM, with the encoder being the same in both cases; the training parameters for this stage is provided in Table IX.

2) Model Retrieval: The first column of Table VI shows the details of the model architectures and training configurations. For each dataset, we first generate its CAML embedding by (1) averaging the query image embeddings within each class to get class embeddings, (2) concatenating the class embeddings into one long vector, (3) passing the combined class embeddings through two linear layers to produce the final dataset embedding. Next, the dataset embedding is combined with the timestep embedding via a projection layer, and the resulting representation is used as input to the flow matching model.

a) A mini-Imagenet prior for CFM: As mentioned in Sec. III-A0c, we attempted this experiment with priors from a pre-trained mini-Imagenet. There were a few technical hurdles with the implantation of these weights. First, for 1-channel datasets such as MNIST, the input weight shapes are smaller than those of the mini-Imagenet model. Second, the classification head of a mini-Imagenet model predicts a much greater number of classes than our test datasets. Our procedure is as follows: we train a small-CNN model [30] on mini-Imagenet until its accuracy stabilizes. Next, we take the mean μ and standard deviation σ of its classifier head. Using these statistics, we initialized the classifier heads of our base models as $\mathcal{N}(\mu, \sigma^2 \boldsymbol{I})$. For the rest of the base model, we pad to the prior's shape if necessary, and we implant the pre-trained weights directly before adding Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$

TABLE IX
CFM/DDPM

Parameters	AE CFM	AE DDPM
Flow/Diffusion Optimizer	AdamW	AdamW
Flow/Diffusion LR	0.001	0.001
Autoencoder Optimizer	AdamW	AdamW
Num Inference Timesteps	100	[100, 1000]
Autoencoder LR	0.001	0.001
Weight Initialization	Kaiming	Normal
Autoencoder Epochs	[1000, 30000]	[1000, 30000]
CFM/DDPM Epochs	[1000, 30000]	[1000, 30000]
Batch Size	[50, 200]	[50, 200]

noise. Since we flow in latent space, our last step is to apply the VAE to the weights we've constructed.

Figure 2 shows the training curve with our mini-Imagenet prior in blue, and with a Gaussian 0-1 prior in orange. It is striking that the loss decreases much faster, but as seen in Table II, the test accuracies are quite poor. This points to an issue such as overfitting, which is likely caused by latent space capacity. Indeed, with our approach of constructing the prior, we invoke the VAE encoder twice: once to encode the prior and once more to encode the target weights. The target weights were those pre-trained on one of the four datasets in Table II, hence it's expected that the distribution is quite distinct from those pre-trained on mini-Imagenet. Due to the size of our latent space (64, as noted in Table VI), it may be insufficient to encode both distributions. Moreover, the loss objective for encoding the prior is not ideal. As the encoder is invoked in forward passes of the CFM, it only learns how to encode the prior such that CFM loss decreases, as opposed to a reconstruction objective. Hence, future work could look to modify encoder training so as to reconstruct both target weights and weights of the prior.

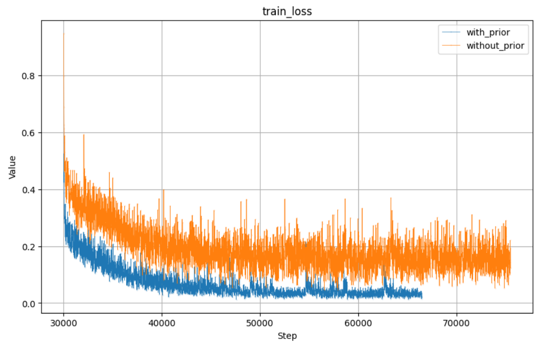

Fig. 2. The training loss curve for the mini-Imagenet run from Table II.

3) Few-Shot Learning: For the few-shot learning experiments, we adopted the same methodology for obtaining dataset embeddings and conditioning parameter generation as in the model retrieval experiment. Classifier heads were trained on 50,000 randomly sampled 5-way 1-shot subsets of the mini-ImageNet dataset [34], and the resulting pre-trained weights were used as training data for the meta-model. The hyperparameter configurations for meta-training are provided in the second column of Table VI. During evaluation, we measured the accuracies of the generated models on two out-of-distribution datasets, CIFAR-10 and STL-10. For each subset, we sampled 50 sets of weights and reported the average accuracy of the top three performing models.

4) Fine-tuning on Out-of-Distribution Data: We provide the hyperparameters of the fine-tuning experiment in Table X.

TABLE X
FINETUNING

Parameters	ResNet18	CNN
Optimizer	AdamW	AdamW
Num Epochs	[50,100]	[50, 100]
Initial LR	1×10^{-2}	1×10^{-5}
Detach Value	0.4	0.4
LR Scheduler	None	CosineAnnealingLR
Minimum LR	1×10^{-2}	5×10^{-7}