# A Flow Matching Algorithm for Many-Shot Adaptation to Unseen Distributions

**Anonymous authors**
Paper under double-blind review

## Abstract

While generative modeling has achieved remarkable success on tasks like natural language-conditioned image generation, enabling model adaptation from example data points remains a relatively underexplored and challenging problem. To this end, we propose Function Projection for Flow Matching (FP-FM), an algorithm that directly conditions generation on samples from the target distribution. FP-FM learns basis functions to span the velocity fields corresponding to a set of training distributions, and adapts to new distributions by computing a simple least-squares projection onto this basis. This enables efficient generation of samples from diverse target distributions without additional training at inference time. We further introduce multiple variants of FP-FM that provide a trade-off in expressivity and compute by enriching the coefficient calculation, e.g., by making the coefficients dependent on time. FP-FM achieves competitive precision and recall relative to baselines across synthetic and image-based datasets, with especially strong gains on unseen distributions, while requiring significantly lower compute.

## 1 Introduction

Generative modeling techniques such as diffusion and flow matching have demonstrated the ability to create high-fidelity, synthetic data modeled after a target distribution (Ho et al., 2020; Lipman et al., 2023; Rombach et al., 2022). Generative modeling is useful in domains where the generated object is the goal, such as image editing or video generation, as well as in domains where the generated object is used for downstream tasks, i.e., acting as a world model for a reinforcement learning agent (Ding et al., 2026; Bruce et al., 2024). For many of these domains, the target distribution changes depending on the user's needs, and there may be limited samples from this target distribution. Furthermore, it is often useful if the synthetic samples are generated with minimal latency. In other words, we want a model capable of efficiently generating *many* distributions based on user specifications.

The most common approach for conditional generation is to include a conditioning variable as an additional input to the model. Most often, this conditioning variable takes the form of natural language (e.g., "an image of a cat") (Ramesh et al., 2022; Saharia et al., 2022), but it may take on other forms such as a one-hot encoding of a class (Ho and Salimans, 2022). However, conditioning variables are not always the best approach. For example, suppose we want to generate images of a specific person. A natural language description of that person's appearance is likely ineffective, while providing images of this person as examples is more intuitive. Conditioning variables are not sufficient for all tasks and we must consider algorithms which take samples from the target distribution as input.

We introduce Function Projection for Flow Matching (FP-FM), an algorithm for conditional generation where the model is conditioned on samples from a target distribution (See Figure 1). FP-FM learns the $k$ most important basis functions in the function space of velocity fields induced by training distributions under flow matching dynamics. By linearly combining these basis functions depending on the provided data, FP-FM is able to approximate the velocity field of a new distribution via a series of least-squares calculations. A concern for generative modeling is the fidelity-overfitting tradeoff, where highly expressive models may only reproduce the provided samples, and insufficiently expressive models generate infeasible samples. To this end, we present three FP-FM variants spanning this spectrum, allowing practitioners to choose the level of model expressivity based on their priorities.

FP-FM is distinguished from prior work based on a few characteristics. Unlike conditional models, FP-FM does not require a variable describing the target distributions; it only requires samples. On the



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

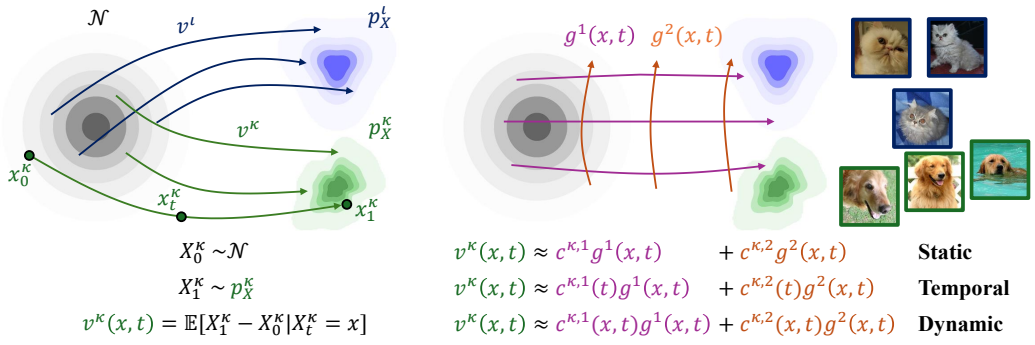


Figure 1: **Conceptual diagram.** (*Left*) Illustration of two distributions, $p_X^\iota$ and $p_X^\kappa$, together with their associated velocity fields $v^\iota$ and $v^\kappa$. Probability densities are depicted as shaded regions, while velocity fields are indicated by arrows. Both stochastic processes share a common initial distribution, a Normal distribution shown in black. (*Right*) FP-FM learns a set of basis functions that span the space of velocity fields, allowing a target field $v^\kappa$ to be represented as a linear combination of these bases. The three FP-FM variants, static, temporal, and dynamic, differ in whether the coefficients are fixed, time-dependent, or state and time-dependent, respectively.

other hand, FP-FM may match unseen target distributions without any training. It is computationally more efficient than finetuning, which requires expensive gradient steps to model a new distribution.

We compare FP-FM against numerous baselines, including unconditional and conditional flow matching, classifier- and distribution-guided methods, and standard finetuning. Across a range of datasets and evaluation settings, we find that FP-FM achieves strong performance on both seen and unseen distributions. In particular, FP-FM consistently improves precision on out-of-distribution targets while maintaining competitive recall, highlighting its ability to capture the desired distribution without over-approximating the data manifold. Moreover, the different variants of FP-FM expose a clear tradeoff between expressivity and computational cost, allowing practitioners to select the most appropriate model for their setting. These results demonstrate that explicitly conditioning on samples using a function space perspective is a practical and effective alternative to existing approaches.

## 2 Background

We first clarify notation. We use $\iota \in \mathcal{I}$ to index over an arbitrary set and $i$ to index over a discrete set. Superscripts denote indices, e.g., $c^\iota$ or $c^i$. When multiple indices are required, we write $c^{\iota,i}$. Subscripts denote time, such as a stochastic process $X_t$, or distributions, such as $p_X$.

### 2.1 Flow Matching

We build upon the flow matching framework (Lipman et al., 2023). Consider a data space $\mathcal{X} \subset \mathbb{R}^n$ and a probability density $p_X : \mathcal{X} \to \mathbb{R}_{>0}$ with $\int_{\mathcal{X}} p_X(x)dx = 1$. We are given $m$ samples $x^1, \ldots, x^m \overset{\text{i.i.d.}}{\sim} p_X$. The objective is to learn a generative model that produces samples $\hat{x}$ whose distribution approximately matches $p_X$. Flow matching defines a stochastic process, $X_t$ for $t \in [0,1]$, such that simulating this stochastic process approximately yields samples from $p_X$. To do so, choose a noise distribution that is easy to sample from, typically the standard normal distribution. Then define $X_0 \sim \mathcal{N}(0, I)$, $X_1 \sim p_X$, and $X_t := (1-t)X_0 + tX_1$. To approximate this process, train a model $v$ to fit the expected velocity field, $v(x,t) = \mathbb{E}[X_1 - X_0 | X_t = x]$, and define the approximated stochastic process as $X_0' \sim \mathcal{N}(0, I)$, $X_t' := X_0' + \int_0^t v(X_\tau', \tau)d\tau$. Then $X_t' \overset{d}{=} X_t$ and it is feasible to simulate $X_t'$, which generates samples from $p_X$.

### 2.2 Function Encoders

In this section, we introduce the function encoder (FE) algorithm (Ingebrand et al., 2025). Consider a Hilbert space $\mathcal{H} = \{f : \mathcal{X} \to \mathbb{R}^n\}$ with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. As an example, an inner product

under an input data distribution $p_x$ is $\langle f, g \rangle_{p_X} := \frac{1}{n}\mathbb{E}_X[f(X)^\top g(X)]$. In particular, this inner product has connections to mean squared error:

$$\|f - \hat{f}\|_{p_X}^2 = \mathbb{E}_X\big[\frac{1}{n}\sum_{i=1}^{n}(f^i(X) - \hat{f}^i(X))^2\big]. \tag{1}$$

Thus, mean squared error is equivalent to squared distance in this function space. The function encoder (Ingebrand et al., 2025) is an algorithm to learn a set of basis functions $\{g^i : \mathcal{X} \to \mathbb{R}^n\}_{i=1}^k$ parametrized as neural networks to span a function space, where any function $f \in \mathcal{H}$ is represented as a linear combination of the basis, $f = \sum_{i=1}^k c^i g^i$. Computing the coefficients $c \in \mathbb{R}^k$ requires a dataset of input-output pairs, $\{(x^j, y^j)\}_{j=1}^m$. These input-output pairs are used to approximate the inner product, and the coefficients are calculated via regularized least squares:

$$c = \left(\begin{bmatrix} \langle g^1, g^1\rangle_{\mathcal{H}} & \dots & \langle g^1, g^k\rangle_{\mathcal{H}} \\ \vdots & \ddots & \vdots \\ \langle g^k, g^1\rangle_{\mathcal{H}} & \dots & \langle g^k, g^k\rangle_{\mathcal{H}} \end{bmatrix} + \lambda I\right)^{-1} \begin{bmatrix} \langle f, g^1\rangle_{\mathcal{H}} \\ \vdots \\ \langle f, g^k\rangle_{\mathcal{H}} \end{bmatrix}. \tag{2}$$

Function encoders are useful because they allow for efficient adaptation to new functions while remaining highly expressive. For more information on this algorithm, see Ingebrand et al. (2025).

# 3 METHODS

## 3.1 PROBLEM SETTING

In this section, we extend the flow matching formulation to a *collection* of distributions. Rather than a single target distribution, we are given a finite family of training distributions, each observed through samples. Our goal is to learn a model that can generate samples from any of these distributions. Moreover, the goal is adaptation: after training, given samples from a previously unseen distribution, the model should efficiently adjust and generate samples from this new distribution.

To formalize this setting, let $\{p_X^\iota\}_{\iota \in \mathcal{I}}$ be a set of target data distributions. Assume we only observe a finite subset of the distributions in $\mathcal{I}$ during training. That is, there exists a training subset $\mathcal{T} \subset \mathcal{I}$. We denote a dataset of independent samples from $p_X^\iota$ by $\mathcal{D}^\iota := \{x^{\iota,i}\}_{i=1}^m$, and assume such a dataset is available for each $\iota \in \mathcal{T}$; i.e., we have access to $\{\mathcal{D}^\iota\}_{\iota \in \mathcal{T}}$. The goal is to train a model that can generate samples from each training distribution $p_X^\iota$. After training, our model should generalize to a new distribution $p_X^\iota$, $\iota \in \mathcal{I} \setminus \mathcal{T}$, given an unseen set of samples $x^{\iota,1}, ..., x^{\iota,m} \overset{i.i.d.}{\sim} p_X^\iota$. This implies our learned model conditions its output distribution on the samples $x^{\iota,1}, ..., x^{\iota,m}$. As an example, the most obvious solution to this problem is to finetune a model based on this new dataset.

## 3.2 APPROACH

We begin by highlighting three observations that suggest a function encoder-style algorithm is well-suited to this problem. Firstly, for each $p_X^\iota$, we may define a stochastic process $X_t^\iota$ and its corresponding velocity field $v^\iota$ as in Section 2.1. Thus, we may view this problem as fitting velocity fields in the set $\mathcal{V} = \{v^\iota : \mathcal{X} \times [0,1] \to \mathbb{R}^n | \iota \in \mathcal{I}\}$. Secondly, a crucial detail in the flow matching framework is that the loss function must be mean squared error in order to learn $\mathbb{E}[X_1 - X_0 | X_t = x]$. Mean squared error, in turn, can be viewed as the (squared) norm induced by the distribution-weighted inner product, as highlighted in section 2.2. Thus, it is natural to equip $\mathcal{V}$ with the distribution-weighted inner product as this maintains the geometry present in the flow matching framework. Thirdly, access to the dataset $\mathcal{D}^\iota$ allows us to approximate inner products involving the unobserved $v^\iota$ and a known function $g$:

$$\langle v^\iota, g\rangle_{p_{t,X_t^\iota}} = \mathbb{E}_{t,X_t^\iota}\left[v^\iota(X_t^\iota, t)^\top g(X_t^\iota, t)\right] \tag{3}$$

$$= \mathbb{E}_{t,X_t^\iota}\left[\mathbb{E}_{X_1,X_0|X_t}\left[X_1^\iota - X_0^\iota \mid X_t^\iota\right]^\top g(X_t^\iota, t)\right] \tag{4}$$

$$= \mathbb{E}_{t,X_1^\iota,X_0^\iota}\left[(X_1^\iota - X_0^\iota)^\top g((1-t)X_0^\iota + tX_1^\iota, t)\right] \tag{5}$$

Equation (5) allows us to avoid sampling $X_t^\iota$ directly and does not require access to $v^\iota$. Therefore, we may approximate inner products involving $v^\iota$ under the current data assumptions. With these three observations taken together, it is clear this setting fits neatly into the function encoder framework. In the following sections, we will introduce three variations of our method building upon these ideas.

### 3.2.1 STATIC FP-FM

As suggested by the previous section, the most obvious approach is to directly apply the function encoder algorithm to this setting. That is, we approximate the velocity field $v^\iota$ as

$$v^\iota(x,t) \approx \sum_{i=1}^{k} c^{\iota,i} g^i(x,t), \tag{6}$$

where $\{g^i\}_{i=1}^k$ are neural networks and $c^\iota \in \mathbb{R}^k$. As the notation implies, for each velocity field $v^\iota$, there is a corresponding vector $c^\iota$. We may calculate the coefficients using least squares as above:

$$c^\iota = \left( \begin{bmatrix} \langle g^1, g^1 \rangle_{p_{t,X_t^\iota}} & \cdots & \langle g^1, g^k \rangle_{p_{t,X_t^\iota}} \\ \vdots & \ddots & \vdots \\ \langle g^k, g^1 \rangle_{p_{t,X_t^\iota}} & \cdots & \langle g^k, g^k \rangle_{p_{t,X_t^\iota}} \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \langle v^\iota, g^1 \rangle_{p_{t,X_t^\iota}} \\ \vdots \\ \langle v^\iota, g^k \rangle_{p_{t,X_t^\iota}} \end{bmatrix}. \tag{7}$$

The key detail is that the distribution used to compute the inner product is $p_{t,X_t^\iota}$, the distribution of $X_t$'s corresponding to a specific target distribution $p_X^\iota$. Interestingly, this implies there is a separate inner product for each target distribution. This poses interesting questions from a mathematical perspective, which we discuss in Appendix B. For the purposes of flow matching, the key detail is that we want one set of basis functions to span all of the target velocity fields, but we use a distribution-specific inner product to find the best approximation for a given velocity field within this basis. To train the basis functions, we leverage the function encoder algorithm (Ingebrand et al., 2025), where a different inner product is used to compute each $c^\iota$. See Appendix G.

We refer to this variant of the algorithm as `Static FP-FM`. So long as we use a sufficient number of sufficiently expressive basis functions (see Ingebrand et al. (2025)), we may reasonably approximate the velocity fields of the training functions, $\{v^\iota | \iota \in \mathcal{T}\}$. Thus, the constant function encoder may generate new samples that are distributed approximately the same as any distribution $p_X^\iota$ in its training set. However, for $p_X^\iota, \iota \in \mathcal{I} \setminus \mathcal{T}$, problems arise. Using Equation (7) yields the best approximation of any new velocity field $v_\iota$ within the linear span of the basis. However, the continuity equation is nonlinear, that is, the relationship between a distribution $p_X^\iota$ and its velocity field $v^\iota$ is nonlinear. Therefore, even if the basis spans the velocity fields in the training set, an "easy" distribution such as a mixture is not necessarily within the span of the basis. In the following sections, we will introduce modifications to this approach that address this issue.

### 3.2.2 TEMPORAL FP-FM

To make FP-FM transfer to new distributions, we must correct the nonlinearity issue. The approximation of a given velocity field depends on two parts: The basis functions and the coefficients. The basis functions themselves are already highly expressive, so instead we modify the coefficient calculation. A straightforward way of doing so is to make the coefficients dependent on time. Let us first introduce a variation of the notation. For a given time $t$, and any function $h : \mathcal{X} \times [0,1] \to \mathbb{R}^n$, let the function at a fixed time $t$ be denoted as $h_t(\cdot) := h(\cdot, t)$. Then we can modify the equations above:

$$v_t^\iota(x) \approx \sum_{i=1}^{k} c^{\iota,i}(t) g_t^i(x). \tag{8}$$

To calculate the coefficients, we must also change the inner product:

$$c^\iota(t) = \left( \begin{bmatrix} \langle g_t^1, g_t^1 \rangle_{p_{X_t^\iota|t}} & \cdots & \langle g_t^1, g_t^k \rangle_{p_{X_t^\iota|t}} \\ \vdots & \ddots & \vdots \\ \langle g_t^k, g_t^1 \rangle_{p_{X_t^\iota|t}} & \cdots & \langle g_t^k, g_t^k \rangle_{p_{X_t^\iota|t}} \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \langle v_t^\iota, g_t^1 \rangle_{p_{X_t^\iota|t}} \\ \vdots \\ \langle v_t^\iota, g_t^k \rangle_{p_{X_t^\iota|t}} \end{bmatrix}. \tag{9}$$

Note that the distribution for the inner product is $p_{X_t^\iota|t}$; that is, it depends on the distribution of states at that specific time $t$. While this change appears small in notation, its impact is profound. We offer

three separate ways to understand the significance, starting with the concrete and moving towards the abstract. Firstly, during generation, we must calculate the coefficients at every time $t$, rather than only once. Secondly, we can now view the coefficients as a function $c^\iota : [0,1] \to \mathbb{R}^k$, meaning that the coefficients are now a function of time. Thirdly, we can view this as modifying the function space definition. Where before we were training basis functions to span $\{v^\iota : \mathcal{X} \times [0,1] \to \mathbb{R}^n | \iota \in \mathcal{I}\}$, we are now training basis functions to span $\{v_t^\iota : \mathcal{X} \to \mathbb{R}^n | \iota \in \mathcal{I}, t \in [0,1]\}$. Where before we had $k$ basis functions $\{g^i\}_{i=1}^k$, we now have $k$ basis functions at every time $t$, i.e., $\{g_t^i\}_{i=1}^k$.

### 3.2.3 DYNAMIC FP-FM

In this section, we take this idea to its natural conclusion. If we can make the model more expressive by making its coefficients depend on time, then we can further increase the expressivity by making the coefficients depend on the state:

$$v^\iota(x,t) \approx \sum_{i=1}^{k} c^{\iota,i}(x,t) g^i(x,t). \tag{10}$$

To compute the localized coefficients $c^\iota(x,t)$, we evaluate the inner product defined previously, but condition it on the observed state $X_t = x$ and time $t$. Denoting this conditional inner product as $\langle \cdot, \cdot \rangle_{x,t}$, the coefficients are computed as the solution to the localized least-squares problem:

$$c^\iota(x,t) = \left( \begin{bmatrix} \langle g^1, g^1 \rangle_{x,t} & \dots & \langle g^1, g^k \rangle_{x,t} \\ \vdots & \ddots & \vdots \\ \langle g^k, g^1 \rangle_{x,t} & \dots & \langle g^k, g^k \rangle_{x,t} \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \langle v^\iota, g^1 \rangle_{x,t} \\ \vdots \\ \langle v^\iota, g^k \rangle_{x,t} \end{bmatrix}. \tag{11}$$

Because $x$ and $t$ are fixed, the conditional expectation collapses. Thus, we can directly compute the left-hand side as $\langle g^i, g^j \rangle_{x,t} = \frac{1}{n} g^i(x,t)^\top g^j(x,t)$. However, we still do not have direct access to the true velocity field $v^\iota$, meaning we must still use Equation (4). By exploiting the conditional distribution, the right-hand side simplifies to $\langle v^\iota, g^i \rangle_{x,t} = \frac{1}{n} \mathbb{E}_{X_1, X_0 | X_t} \left[ X_1^\iota - X_0^\iota \mid X_t^\iota = x \right]^\top g^i(x,t)$, which we again approximate from empirical data. Putting these two facts together, solving Equation (11) effectively performs two steps: 1) computing an empirical estimate of the mean velocity field given $X_t^\iota = x$, and 2) finding the best approximation of this vector within the $k$-dimensional subspace of $\mathbb{R}^n$ spanned by the basis $\{g^i(x,t)\}_{i=1}^k$. We can interpret this as similar to a purely data-driven approach, except the learned functions $\{g^i\}$ play the crucial role of regularizing the approximation based on the data present in the training set.

Another important consideration is approximating the conditional expectation $\mathbb{E}[X_1^\iota - X_0^\iota \mid X_t^\iota = x]$. Because this expectation is conditioned on the fixed state, we can no longer estimate it by drawing independent samples of $X_0^\iota$ and $X_1^\iota$; any valid pair must strictly satisfy the deterministic constraint $X_t^\iota = (1-t)X_0^\iota + tX_1^\iota$. Instead, we bypass this issue using the following general result:

**Theorem 1.** *Given the fixed noisy example $X_t = x$ and a random variable $X_1$, let $X_0^* = \frac{x - tX_1}{1-t}$. The conditional expectation can be expressed as an expectation over the marginal distribution of $X_1$:*

$$\mathbb{E}[X_1 - X_0 \mid X_t = x] = \mathbb{E}_{X_1} \left[ (X_1 - X_0^*) \frac{p(X_0^*)}{\mathbb{E}_{X_1'}[p(X_0^{*\prime})]} \right].$$

This formulation is highly amenable to Monte Carlo approximation. We already possess samples from the target distribution $p(X_1)$, and computing $X_0^*$ is a trivial linear operation given $x$ and $X_1$. Furthermore, assuming the base noise distribution is Gaussian, $p(X_0^*)$ can be evaluated analytically. Together, these properties yield an efficient, self-normalized importance sampling estimator for the conditional expectation. Furthermore, the basis functions play the crucial role of preventing this estimator from overfitting to a single sample, since high-dimensional Gaussian importance weights tend to collapse when $t \approx 1$. Because the expectation is projected onto the basis, it cannot simply move towards the most likely sample directly; it instead must move in a direction which is generally good for generating images, as stored in the basis functions. A full derivation of the theorem and algorithmic details regarding the sampling procedure are deferred to Appendix A.

### 3.2.4 TRADEOFFS

Next, we will discuss the tradeoffs between these three approaches. As suggested in the preceding sections, we observe an ordering in expressive power: `Dynamic FP-FM` exhibits the greatest

expressivity, followed by the `Temporal FP-FM`, and finally `Static FP-FM`. On the other hand, we observe the opposite relationship in terms of compute. `Static FP-FM` is the most compute efficient, as it computes the coefficients only once per distribution. `Temporal FP-FM` and `Dynamic FP-FM` require more compute, albeit with different scaling properties. `Temporal FP-FM` requires less least-squares solves, but more forward passes than `Dynamic FP-FM`, since it needs to forward pass the basis at the current $\hat{x}_t$ and also at the empirical $x_t$'s. Meanwhile, `Dynamic FP-FM` requires a least-squares solve for every time and sample, whereas `Temporal FP-FM` only requires one least-squares solve at each time. In the following sections, we will empirically compare the three approaches to validate these tradeoffs.

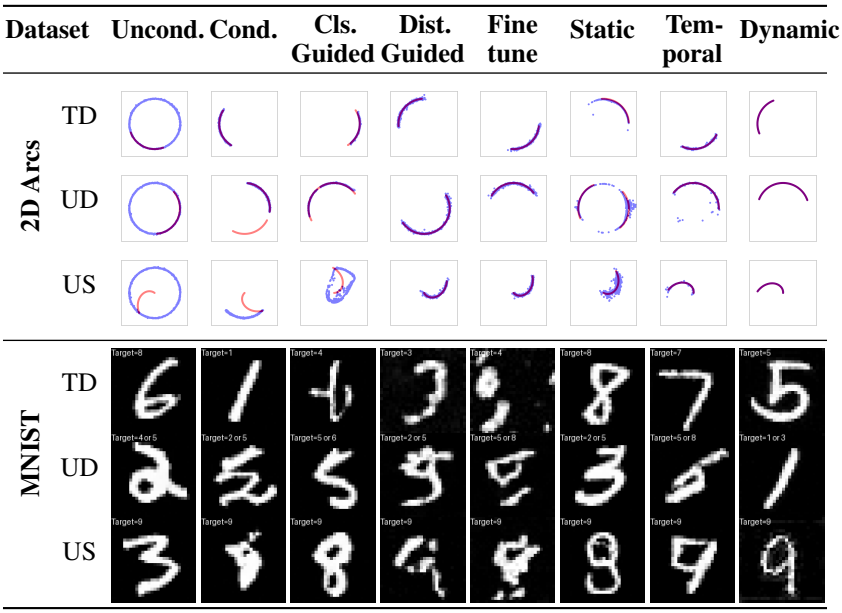


Figure 2: Qualitative comparison across datasets and distribution types. For 2D Arcs, the image shows the full distributions, where red points correspond to samples from the target distribution and blue points are samples from the model. For MNIST, each image shows one sample from the model.

## 4 Experiments

We validate our approach on three datasets: *2D Arcs*, *MNIST* (LeCun et al., 1998), and *ImageNet* (Russakovsky et al., 2015). 2D Arcs is a low-dimensional dataset that enables visualization of the distributions. MNIST evaluates performance on structured image data, while ImageNet tests scalability to large-scale, high-dimensional data. To measure the generalization capabilities of these approaches, we evaluate on three distribution splits for each dataset. *Training Distribution (TD)* is a distribution that was seen during training. *Unseen Distribution (UD)* is a distribution that is not in the training set, but the samples have been seen. For example, if we train on MNIST on digits 0 through 8, an unseen distribution could be a mixture of 0 and 1. *Unseen Support (US)* is an unseen distribution with a different support than what was seen during training. To continue the analogy, if we train on MNIST 0 through 8, the distribution corresponding to 9 has an unseen support.

We compare against six baselines. `Unconditional` is a flow matching model (Lipman et al., 2023) trained on all training distributions as if they were a single distribution. This acts as naive baseline, and as a starting point for some of the other algorithms. `Conditional` is a model provided with a conditioning variable to describe the target distribution, e.g., a one-hot encoding of the class (Ho and Salimans, 2022). `Classifier-Guided` trains a classifier for a given target distribution, then uses the score to guide generation of a pretrained, unconditional model (Dhariwal and Nichol, 2021). `Distribution-Guided` starts with a pretrained, unconditional model. It then trains a secondary model to produce noise inputs that, when passed through the pretrained model, generate samples from the target distribution. See Appendix C for a visualization. `Finetune` is the standard approach. Given a pretrained unconditional model, finetune it on the data provided for the target distribution.

| Algorithm | Precision ↑ TD | UD | US | Recall ↑ TD | UD | US | Time (s) ↓ |
|---|---|---|---|---|---|---|---|
| `Unconditional` | 0.108 | 0.291 | 0.003 | 0.997 | 0.999 | 0.032 | 0.17 |
| `Conditional` | 0.369 | 0.608 | 0.006 | 0.998 | 0.613 | 0.041 | 0.17 |
| `Classifier-Guided` | 0.407 | 0.654 | 0.009 | 0.836 | 0.867 | 0.134 | 7.08 |
| `Distribution-Guided` | 0.980 | 0.980 | 0.016 | 0.999 | 0.999 | 0.997 | 16.48 |
| `Finetune` | 0.612 | 0.735 | 0.385 | 0.998 | 0.999 | 0.999 | 4.77 |
| `Static FP-FM` | 0.324 | 0.320 | 0.086 | 0.847 | 0.811 | 0.894 | 0.21 |
| `Temporal FP-FM` | 0.586 | 0.763 | 0.709 | 0.997 | 0.997 | 0.998 | 0.72 |
| `Dynamic FP-FM` | 0.991 | 0.997 | 0.934 | 0.989 | 0.985 | 0.972 | 10.99 |
| **Context** | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | - |

Table 1: **2D Arc.** Comparison of methods across TD, UD, and US settings. Arrows indicate whether higher (↑) or lower (↓) is better. Results are averaged over 5 seeds.

| Algorithm | Precision ↑ TD | UD | US | Recall ↑ TD | UD | US | FID ↓ TD | UD | US | Time ↓ |
|---|---|---|---|---|---|---|---|---|---|---|
| `Unconditional` | 0.342 | 0.464 | 0.303 | 0.705 | 0.705 | 0.666 | 97.93 | 88.59 | 92.84 | 10.31 |
| `Conditional` | 0.875 | 0.763 | 0.309 | 0.877 | 0.759 | 0.783 | 12.54 | 23.18 | 31.92 | 10.78 |
| `Classifier-Guided` | 0.582 | 0.632 | 0.440 | 0.719 | 0.720 | 0.714 | 82.00 | 80.18 | 88.98 | 29.19 |
| `Distribution-Guided` | 0.753 | 0.725 | 0.686 | 0.781 | 0.760 | 0.734 | 18.39 | 21.99 | 15.71 | 626.17 |
| `Finetune` | 0.728 | 0.728 | 0.737 | 0.590 | 0.606 | 0.566 | 80.31 | 81.37 | 81.45 | 215.71 |
| `Static FP-FM` | 0.783 | 0.757 | 0.554 | 0.865 | 0.856 | 0.766 | 16.71 | 21.03 | 22.53 | 10.63 |
| `Temporal FP-FM` | 0.771 | 0.767 | 0.569 | 0.885 | 0.897 | 0.797 | 14.13 | 17.49 | 17.69 | 20.78 |
| `Dynamic FP-FM` | 0.759 | 0.829 | 0.723 | 0.818 | 0.817 | 0.837 | 17.38 | 18.38 | 18.37 | 21.96 |
| **Context** | 0.891 | 0.898 | 0.900 | 0.897 | 0.893 | 0.903 | 11.16 | 12.91 | 10.27 | - |

Table 2: **MNIST.** Comparison of methods across TD, UD, and US settings. Arrows indicate whether higher (↑) or lower (↓) is better. Results are averaged over 5 seeds.

Note that there are many tradeoffs you can make during finetuning; e.g., LoRA (Hu et al., 2021) is more compute efficient but less expressive. **Context** is an oracle baseline consisting of a small set of real images compared against a different set of real images, and acts as an approximate optimal performance. For more detailed descriptions of the baselines, see Appendix C.

To evaluate these approaches, we use various metrics: *Time* measures how long it takes to generate a batch of samples. *Precision* measures how well the generated samples match the target distribution; *Recall* measures how well real samples match the generated distribution (Kynkäänniemi et al., 2019). For the image-based datasets, we use *FID* (Heusel et al., 2017). For implementation details, see Appendix C. Results are shown in Figure 2 and in Tables 1, 2, and 3.

### 4.1 Results

**2D Arcs.** The TDs are one quarter of the unit circle by central angle, the UDs are mixtures of two training distributions, and the US is a spiral from the origin to the unit circle. Each baseline is trained for 10,000 gradient steps; see Figure 2. For brevity, standard deviations are in the appendix.

Qualitatively, `Static FP-FM` fits TDs but fails on UD and US, while `Temporal FP-FM` generalizes to all three settings and `Dynamic FP-FM` produces the most accurate reconstructions. Among the baselines, `Unconditional` fails to capture specific targets, `Conditional` and `Classifier-Guided` do not generalize to novel supports, and `Distribution-Guided` and `Finetune` approximate all settings with noticeable error. Consistent with these observations, many methods achieve high recall but low precision (especially on US), indicating overestimated support, whereas `Dynamic FP-FM` achieves the highest precision on UD and US. `Temporal FP-FM` also generalizes well to US, but is not as precise as `Dynamic FP-FM`.

**MNIST.** The TDs consist of digit classes 0 through 8. UDs are defined as mixtures of any two of these classes, while US corresponds to the unseen class 9. We train each baseline for 20,000 gradient steps. Qualitatively, the trends in 2D Arcs carry over to MNIST. `Static FP-FM` successfully

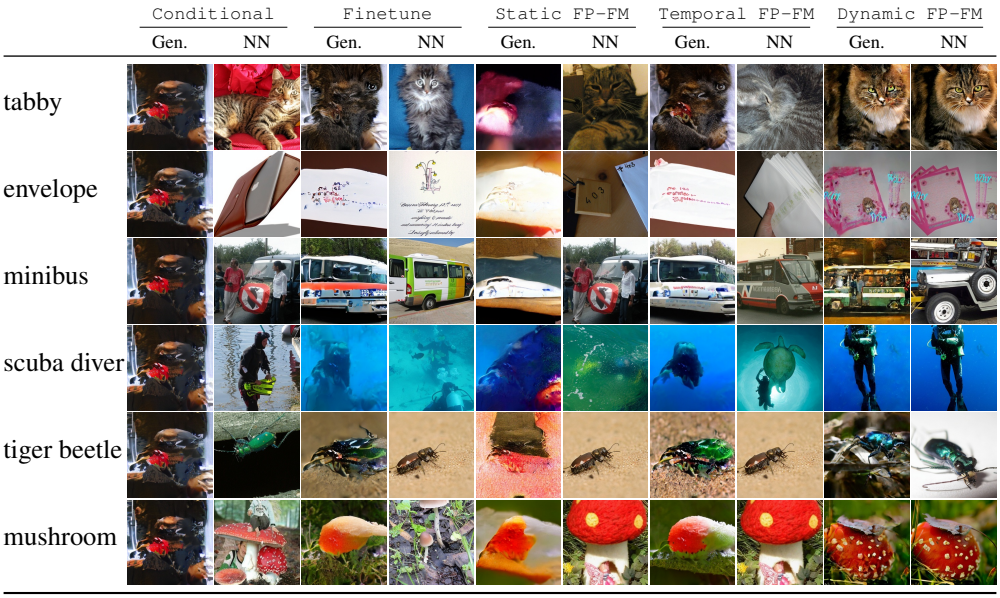


Figure 3: Generated samples and their nearest neighbors in the context set. Within each method cell, the generated sample is shown on the left and its nearest neighbor is shown on the right. The same noise $x_0$ is used for all models. Nearest neighbors are calculated in embedding space. The classes are heldout. This causes `Conditional` to fail, since the class variable is unseen.

| Algorithm | Precision ↑ TD | Precision ↑ US | Recall ↑ TD | Recall ↑ US | FID ↓ TD | FID ↓ US | Time (s) ↓ |
|---|---|---|---|---|---|---|---|
| `Conditional` | 0.478 | 0.011 | 0.181 | 0.157 | 154.1 | 382.2 | 19.6 |
| `Finetune` (LoRA) | 0.487 | 0.264 | 0.192 | 0.261 | 150.7 | 222.4 | 607.5 |
| `Static FP-FM` | 0.023 | 0.018 | 0.194 | 0.238 | 348.1 | 339.8 | 10.7 |
| `Temporal FP-FM` | 0.289 | 0.252 | 0.335 | 0.354 | 207.6 | 212.8 | 20.8 |
| `Dynamic FP-FM` | 0.412 | 0.433 | 0.541 | 0.545 | 136.6 | 130.2 | 14.1 |
| **Context** | 0.696 | 0.693 | 0.702 | 0.676 | 75.8 | 76.4 | – |

Table 3: **Imagenet.** Comparison of methods across TD and US settings. Arrows indicate whether higher (↑) or lower (↓) is better. Time is measured for generating 128 samples.

models TDs but fails to generalize to UD and US. `Temporal FP-FM` improves generalization to mixtures and unseen classes, producing reasonable samples across all settings. `Dynamic FP-FM` provides the most consistent performance, performing especially well on UD and US. These trends are reflected in the quantitative metrics. Other methods tend to achieve reasonable recall but significantly lower precision and worse FID scores. `Distribution-Guided` has high precision even on US, but this comes at the cost of a significantly increased generation time. We also use MNIST to perform ablations on the number of shots and the number of basis functions. See Appendix E.

**ImageNet.** Similar to MNIST, we use a random subset with 900 classes out of 1000 to form the TDs, and the rest are used for US. We use a latent Diffusion Transformer (DiT) (Dosovitskiy et al., 2021; Peebles and Xie, 2022) backbone and each baseline is trained for 150,000 training steps. Due to the expense of training, we only train one model and focus on key baselines. See Figure 3 and Table 3. Quantitatively, `Dynamic FP-FM` achieves the highest precision and recall on US, as expected. Interestingly, `Temporal FP-FM` performs comparably to `Finetune` despite requiring roughly 30$\times$ less compute. Qualitatively, `Dynamic FP-FM` largely reconstructs the images provided in context, introducing only minor visual perturbations. A particularly striking pattern emerges between `Temporal FP-FM` and `Finetune`: across all six classes, the two methods generate highly similar images. Because all methods are initialized from the same random noise, this similarity suggests that `Temporal FP-FM` and `Finetune` learn closely aligned conditional expectations. These results indicate that `Temporal FP-FM` may provide a computationally efficient alternative to finetuning while producing comparable samples.

## 5 Discussion

Overall, the experiments show that `Dynamic FP-FM` achieves the best precision, followed by `Temporal FP-FM` and then `Static FP-FM`. However, `Dynamic FP-FM` tends to generate samples similar to the context, which limits its usefulness in many image-based settings. Nonetheless, it may find uses in other settings where only minor perturbations are needed. On the hand, `Temporal FP-FM` performs comparably to `Finetune` despite having much lower cost. We therefore suggests practitioners should consider `Temporal FP-FM` for image-based in-context generation.

**Future work.** One consideration is the desired expressivity of the model. To that end, an direction is to generalize our method to a spectrum of methods with controllable expressivity. For example, if the coefficients are thought of as a function of state and time, then imposing characteristics such as Lipschitz constraints control the expressivity of the model. In this sense, `Static FP-FM`'s coefficients are 0-Lipschitz with respect to state and time, while `Dynamic FP-FM`'s coefficients have no Lipschitz constraints. `Temporal FP-FM` is 0-Lipschitz with respect to state but not time. Generalizing this method to a spectrum is an direction, and we leave it to future work.

**Limitations.** While FP-FM is significantly more computationally efficient than finetuning, it is slightly more expensive than a `Conditional` model, as shown in the experiments. This tradeoff is empirically justified by improvements in UD and US performance. FP-FM also requires samples, rather than a conditioning variable. This assumption may not be reasonable in every setting. Finally, performance improves with the number and quality of samples provided. In extremely low-sample regimes, the method may struggle to accurately capture the underlying manifold, and the required number of samples depends on the target distribution.

## 6 Related Work

FP-FM builds on parametric generative modeling (Ho et al., 2020; Song et al., 2021) and the ODE viewpoint underlying flow and rectified-flow models (Lipman et al., 2023; Liu et al., 2023), in which a velocity field transports noise to a target distribution via a neural ODE (Chen et al., 2018). Unlike class-conditioned or text-conditioned models (Winkler et al., 2019; Ho and Salimans, 2022; Ramesh et al., 2022; Saharia et al., 2022; Rombach et al., 2022; Balaji et al., 2022), FP-FM conditions directly on samples, avoiding conditioning variables that may require extensive data or inadequately describe the target. It also adapts without the parameter updates used by fine-tuning methods (Ruiz et al., 2023; Dosovitskiy et al., 2021) or the additional training required by classifier-guided and Langevin-based approaches (Dhariwal and Nichol, 2021; Song and Ermon, 2019).

FP-FM can also be viewed as meta-learning (Finn et al., 2017), with velocity-field approximation as the inner problem and basis-function training as the outer problem. Unlike methods that encode target functions through learned latent representations (Garnelo et al., 2018; Edwards and Storkey, 2017), FP-FM computes representations by least squares. Related basis function methods (Wilson et al., 2015; Bertinetto et al., 2019) operate in kernel space and use the Woodbury identity, causing their solves to scale poorly with the amount of data. FP-FM instead operates in feature space and scales efficiently. The closest work, FE-based neural ODEs (Ingebrand et al., 2024), learns basis functions from discrete states of deterministic systems with unobserved derivatives, whereas FP-FM directly fits velocity fields under different data assumptions.

## 7 Conclusion

We introduced FP-FM, a flow matching algorithm that enables rapid adaptation to unseen target distributions. By reframing velocity field estimation in terms of a distribution-weighted inner product, we showed how Hilbert spaces naturally integrate with the flow matching objective. The three FP-FM variants progressively increase expressivity by varying the coefficient calculation, with more expressive models fitting unseen distributions more accurately. This opens several directions for future work, such as more fine-grained control of the expressivity. Overall, FP-FM is a natural first step towards adapting generative models to unseen distributions.

### Ethics statement

FP-FM may lower the barrier to producing highly targeted synthetic data, potentially enabling impersonation, data poisoning, or the amplification of biased or misleading content. If an industry-grade model is made publicly available, standard security safeguards should be implemented to prevent malicious use.

### AI Use Statement

In this work, we used generative AI for tasks such as writing code and editing the text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### Reproducibility Statement

Implementation details are described in the appendix. A large set of ImageNet generated samples are attached in the supplementary material. Code will be released upon publication.

## References


Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*. OpenReview.net, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10674–10685. IEEE, 2022.

Jingtao Ding, Yunke Zhang, Yu Shang, Yuheng Zhang, Zefang Zong, Jie Feng, Yuan Yuan, Hongyuan Su, Nian Li, Nicholas Sukiennik, Fengli Xu, and Yong Li. Understanding world or predicting future? A comprehensive survey of world models. *ACM Comput. Surv.*, 58(3):57:1–57:38, 2026.

Jake Bruce, Michael D. Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal M. P. Behbahani, Stephanie C. Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott E. Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments. In *ICML*, Proceedings of Machine Learning Research, pages 4603–4623. PMLR / OpenReview.net, 2024.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *CoRR*, abs/2204.06125, 2022.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, 2022.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022.

Tyler Ingebrand, Adam J. Thorpe, and Ufuk Topcu. Function encoders: A principled approach to transfer learning in hilbert spaces. In *ICML*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, pages 8780–8794, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *NeurIPS*, pages 3929–3938, 2019.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, pages 6626–6637, 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021.

William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*. OpenReview.net, 2021.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*. OpenReview.net, 2023.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2018.

Christina Winkler, Daniel E. Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *CoRR*, abs/1912.00042, 2019.

Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers. *CoRR*, abs/2211.01324, 2022.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *CVPR*, pages 22500–22510. IEEE, 2023.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, pages 11895–11907, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, Proceedings of Machine Learning Research, pages 1126–1135. PMLR, 2017.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Jimenez Rezende, and S. M. Ali Eslami. Conditional neural processes. In *ICML*, Proceedings of Machine Learning Research, pages 1690–1699. PMLR, 2018.

Harrison Edwards and Amos J. Storkey. Towards a neural statistician. In *ICLR*, 2017.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. *CoRR*, abs/1511.02222, 2015.

Luca Bertinetto, João F. Henriques, Philip H. S. Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *ICLR (Poster)*. OpenReview.net, 2019.

Tyler Ingebrand, Adam J. Thorpe, and Ufuk Topcu. Zero-shot transfer of neural odes. In *NeurIPS*, 2024.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI (3)*, Lecture Notes in Computer Science, pages 234–241. Springer, 2015.

Su Ann Low, Quentin Rommel, Kevin S. Miller, Adam J. Thorpe, and Ufuk Topcu. Function spaces without kernels: Learning compact hilbert space representations, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025.

## A Derivation of Theorem 1

In this section, we show the derivation for Theorem 1, and additionally provide a diagram which makes the intuition clear. See Figure 4. In the proof, we omit the $\iota$ superscript for conciseness. Note that we make the standard assumptions for this setting: $X_0$ and $X_1$ are independent, $p(x_0)$, $p(x_1)$ exists as densities with respect to the Lebesgue measure, $t < 1$, the noise distribution is Gaussian, and the interpolation is linear.

*Proof.*

$$\mathbb{E}_{X_1,X_0|X_t=x_t}[X_1 - X_0|X_t = x_t] = \int_{\mathcal{X}\times\mathcal{X}} (x_1 - x_0)p(x_1, x_0|x_t)dx_1dx_0 \tag{12}$$

$$= \int_{\mathcal{X}\times\mathcal{X}} (x_1 - x_0)\frac{p(x_t|x_1, x_0)p(x_0, x_1)}{p(x_t)}dx_1dx_0 \tag{13}$$

$$= \int_{\mathcal{X}\times\mathcal{X}} (x_1 - x_0)\frac{p(x_t|x_1, x_0)p(x_0)p(x_1)}{p(x_t)}dx_1dx_0 \tag{14}$$

$$= \int_{\mathcal{X}\times\mathcal{X}} (x_1 - x_0)\frac{\delta\left(x_t - (tx_1 + (1-t)\,x_0)\right)p(x_0)p(x_1)}{p(x_t)}dx_1dx_0 \tag{15}$$

Fix $x_1$ and let $h(x_0) = x_t - (tx_1 + (1-t)\,x_0)$. Then the (only) root $x_0^* = \frac{x_t - tx_1}{1-t}$. Using $\int f(x)\delta\left(h(x)\right)dx = \frac{f(x^*)}{|\det J_h(x^*)|}$, and the fact that $|\det J_h(x_0^*)| = |\det(-(1-t)I)| = |(1-t)^n|$ for our specific choice of $h$,

$$= \int_{\mathcal{X}} (x_1 - x_0^*)\frac{\frac{1}{|\det J_h(x_0^*)|}p(x_0^*)p(x_1)}{p(x_t)}dx_1 \tag{16}$$

$$= \int_{\mathcal{X}} (x_1 - x_0^*)\frac{\frac{1}{|(1-t)^n|}p(x_0^*)p(x_1)}{p(x_t)}dx_1 \tag{17}$$

Using the same trick as before on $p(x_t)$,

$$p(x_t) = \int_{\mathcal{X}\times\mathcal{X}} \delta\left(x_t - (tx_1 + (1-t)\,x_0)\right)p(x_0)p(x_1)dx_1dx_0 \tag{18}$$

$$= \int_{\mathcal{X}} \frac{1}{|(1-t)^n|}p(x_0^*)p(x_1)dx_1. \tag{19}$$

Then substituting back into (17),

$$= \frac{\int_{\mathcal{X}}(x_1 - x_0^*)\frac{1}{|(1-t)^n|}p(x_0^*)p(x_1)dx_1}{\int_{\mathcal{X}}\frac{1}{|(1-t)^n|}p(x_0^*)p(x_1)dx_1} \tag{20}$$

$$= \frac{\int_{\mathcal{X}}(x_1 - x_0^*)p(x_0^*)p(x_1)dx_1}{\int_{\mathcal{X}}p(x_0^*)p(x_1)dx_1} \tag{21}$$

$$= E_{X_1}[(X_1 - X_0^*)\frac{p(X_0^*)}{E_{X_1}[p(X_0^*)]}] \tag{22}$$

□

**Sampling Procedure** To approximate this expectation for a given $x_t^\iota$, we first sample a set of $x_1^\iota$'s. Typically, this consists of the provided samples from the new distribution. Then, for each $x_1^{\iota,i}$, compute the corresponding initial state $x_0^{\iota,i}$ and then $w^{\iota,i} = p(x_0^{\iota,i})$. Normalize the $w^{\iota,i}$'s to account for $E_{X_1}[p(X_0^*)]$ with $\tilde{w}^{\iota,j} = w^{\iota,j} / \sum_{i=1}^m w^{\iota,i}$. Lastly, compute the empirical mean weighted by $\tilde{w}^{\iota,i}$,

$$\mathbb{E}_{X_1,X_0|X_t=x}[X_1 - X_0|X_t = x_t] \approx \sum_{i=1}^m (x_1^{\iota,i} - x_0^{\iota,i})\tilde{w}^{\iota,i} \tag{23}$$

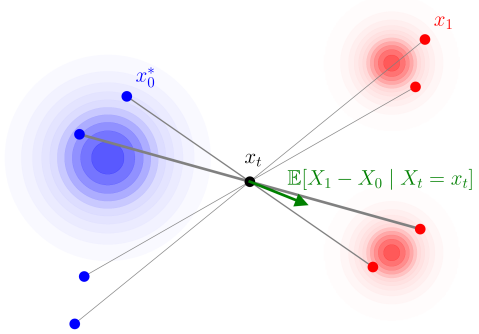


Figure 4: **Visualization of Theorem 1.** Consider a noise distribution (shown in blue) and a mixture of Gaussians target distribution (shown in red). For any given $x_t$ (black point) and set of $x_1$'s (red points), we may compute the corresponding $x_0^*$'s using the equation $x_0^* = \frac{x_t - t x_1}{1-t}$. However, not all $x_0^*$'s are probable. We represent the likelihood of $x_0^*, x_1$ given $x_t$ via the thickness of the line connecting them. Clearly, some pairs are more likely. Therefore, the expected velocity field $\mathbb{E}[X_1 - X_0 | X_t]$ (green) will appear to align with the thicker of these lines. This is exactly why we need to consider the probability of $x_0^*$ when computing the expected velocity field.

## B Function-Distribution Pairs

We begin by generalizing this formulation. Let $\mathcal{F} = \{f : \mathcal{X} \to \mathbb{R}^n \mid f \text{ measurable}\}$, where compact $\mathcal{X}$ is the input space and $\mathbb{R}^n$ is the output space. Suppose we observe a finite subset $\mathcal{T} = \{f^i\}_{i=1}^m \subset \mathcal{F}$, and a corresponding set of densities $\mathcal{P} = \{p_X^i : \mathcal{X} \to [0, \infty) | i \in 1...m\}$. Each density $p_X^i$ induces a Hilbert space $\mathcal{H}^i = L^2(p_X^i)$ with inner product $\langle f, g \rangle_i = \int_{\mathcal{X}} f(x)^\top g(x) p_X^i(x) dx$, and assume $f^i \in \mathcal{H}^i$. Note that for $j \neq i$, it is possible that $f^j \notin \mathcal{H}^i$. In other words, our set of functions may or may not exist in each other's function spaces.

As is typical for the function encoder, our goal is to learn a set of functions $\{g^j\}_{j=1}^k$ such that for all $i$, $f^i \approx \sum_{j=1}^k c^{i,j} g^j$ where $c^i \in \mathbb{R}^k$. This allows us to approximate $f^i$ with a linear combination of our basis functions. As is typical, we compute the coefficients using least squares,

$$\mathbb{R}^k \ni c^i = \begin{bmatrix} \langle g^1, g^1 \rangle_i & \dots & \langle g^1, g^k \rangle_i \\ \vdots & \ddots & \vdots \\ \langle g^k, g^1 \rangle_i & \dots & \langle g^k, g^k \rangle_i \end{bmatrix}^{-1} \begin{bmatrix} \langle f^i, g^1 \rangle_i \\ \vdots \\ \langle f^i, g^k \rangle_i \end{bmatrix}. \tag{24}$$

This requires $\forall i, j g^j \in \mathcal{H}^i$, i.e., the basis functions must be members of all Hilbert spaces. However, since we are approximating the basis functions with neural networks, they are continuous. Since $\mathcal{X}$ is compact and $g^j$ is continuous, then it follows from the extreme value theorem that each $g^j$ is bounded by some maximum value $M_j$. Therefore,

$$||g^j||^2_{L^2(p_X^i)} = \int_{\mathcal{X}} ||g^j(x)||_2^2 p_X^i(x) dx \leq M_j^2 \int_X p_X^i(x) dx = M_j^2 < \infty$$

Therefore, $g^j \in \mathcal{H}^i$. In other words, the basis functions are well-defined regardless of the density function. Thus, we only need to train the basis functions to span $\{f^i\}$, and then we may use them to approximate any such function $f^i$. Note that by using Equation (24), we implicitly assume we want to minimize the approximation error on points that are likely under $p_X^i$, but assumption is quite natural given the correspondence between $f^i$ and $p_X^i$.

One interesting result of this formulation is that two functions $f^i, f^j$ may have the same representation, $c^i = c^j$, even if they differ on a set with measure greater than 0. For example, suppose the support of $p_X^i$ and $p_X^j$ are disjoint; Then it is easy to see how they may have the same coefficient representation.

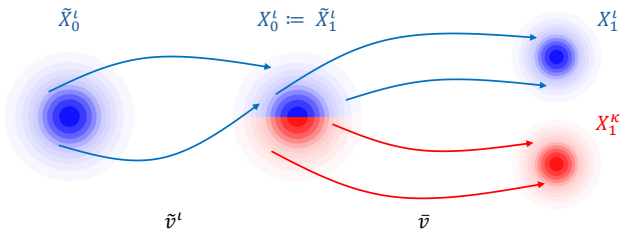


Figure 5: **Visualization of the Distribution-Guided Model.** Suppose we have trained a `Unconditional` model $\bar{v}$ that maps a Normal distribution (middle) to a mixture of two Gaussians (right). The `Unconditional` model maps the top half of its noise distribution to the upper target Gaussian, and the bottom half to the lower target Gaussian. Then, we are interested in generating samples only from the upper Gaussian distribution (i.e., $X_1^\iota$). To do so, train a second velocity field $\tilde{v}^\iota$ which itself maps from a Normal distribution (left) to the upper half of the Normal distribution (middle, blue). Putting these together, a sample is first transitioned from $\tilde{X}_0^\iota$ to $\tilde{X}_1^\iota$ via $\tilde{v}^\iota$, then transitioned from $X_0^\iota := \tilde{X}_1^\iota$ to $X_0^\iota$ via $\bar{v}$. This process yields samples from the target distribution.

This implies we use the same approximation for both functions because this approximation is the most accurate when measured via the corresponding inner product. In terms of flow matching, if the representation for two target distributions were the same, then they would have the same velocity field, and therefore the same sampling distribution. However, since every $X_t^\iota$ has the same noise distribution, the supports of the distributions $p^\iota_{X_t^\iota}$ overlap when $t = 0$, which helps to distinguish between the velocity fields. We have not observed two separate velocity fields having the same coefficients in practice.

## C Baselines and Implementation Details

**Unconditional** is a standard flow matching model trained on all of the training distributions as if they were a single distribution (e.g., a mixture). This acts as naive baseline, and as a starting point for some of the other algorithms. For obvious reasons, this baseline should perform poorly at all of TD, UD, US.

**Conditional** is a model provided with a conditioning variable to describe the target distribution, but is otherwise the same as an `Unconditional` model. For example, for *2D-Arcs*, the conditioning variable is the angle of the center of the arc. While this allows the model to accurate learn the training distributions, it is unclear how this conditioning variable could generalize to a mixture of arcs, or a spiral. For *MNIST*, the conditioning variable is a 10-dimensional vector with 1's for classes within the mixture. Although this conditioning variable can describe mixtures (with multiple ones), it was not trained to do so and so it fails to generalize. Likewise, although it has a one-hot encoding of 9, it has not seen this during training and so it fails to generalize. *ImageNet* uses a similar strategy as MNIST, with classes encoded into a vector. We expect this model to perform well on TD's, but to fail on UD's and US's.

**Classifier-Guided** starts with a pre-trained, `Unconditional` model $\bar{v}$. Then, it trains a classifier $\hat{p}^\iota : \mathcal{X} \times [0,1] \to [0,1]$ to label if a given state-time pair $(x_t, t)$ belongs to the target distribution or not. The score of this classifier is used to guide the `Unconditional` model towards the desired distribution,

$$v^\iota(x_t, t) \approx \bar{v}(x_t, t) + \alpha \nabla_{x_t} \log \hat{p}^\iota(x_t, t),$$

where $\alpha$ is a hyper parameter. The intuition behind this idea is that the `Unconditional` model can generate samples from any of the training distributions, and so we only need to guide the model towards the target distribution using the gradient of the log density as estimated by our model. The main challenge with this method is choosing $\alpha$. To the best of our knowledge, there is not a consistent way of choosing $\alpha$, and it needs to be tuned per dataset and potentially even per distribution. Also, this requires training a classifier per each target distribution, which is similar in expense to finetuning.

We expect this model to perform well on TD and UD. However, it will typically perform poorly on US.

**Distribution-Guided** likewise starts with a pretrained `Unconditional` model $\bar{v}$. It then trains a secondary model $\tilde{v}^\iota$ to produce noise inputs that, when passed through the pretrained model, generate samples from the target distribution. Generating a sample has the following form:

$$\tilde{X}_0^\iota \sim \mathcal{N}(0, I)$$

$$\tilde{X}_t^\iota = \tilde{X}_0^\iota + \int_0^t \tilde{v}^\iota(\tilde{X}_\tau^\iota) d\tau$$

$$X_0^\iota := \tilde{X}_1^\iota$$

$$X_t^\iota = X_0^\iota + \int_0^t \bar{v}(X_\tau^\iota) d\tau$$

Similar to `Classifier-Guided`, the idea is that the `Unconditional` model can already generate any sample in the training distributions. Thus, we only need to figure out which distribution of inputs to the `Unconditional` model will yield the target distribution. Then, we train a separate model to generate this starting distribution. See Figure 5.

While this approach is mathematically elegant, it has numerous practical downsides. First, to train the distribution-specific model, we must first compute the inputs to the `Unconditional` model that generate the target distribution. This involves starting with samples from the target distribution, and integrating backwards in time to recover the initial states. This backwards integration can be numerically unstable. Second, we must train an entirely separate flow matching model, which is expensive. Third, generation is now twice as expensive, since we have two models instead of one. We expect this model to succeed on TD and UD. It should fail on US, since it can only generate samples that the base `Unconditional` model is capable of generating.

**Finetune** is the standard approach. Given a pretrained unconditional model, finetune it on the data provided for the target distribution using the standard flow matching method. This approach can in theory perform well at all of TD, UD, US. However, since we are training then entire model, it may require many samples and a lot of compute to do so.

**Implementation Details.** Across all methods, we align training settings as closely as possible, including architectures, batch sizes, and optimizers. We use an Adam or AdamW optimizer with a learning rate of 1e-3 or 5e-4 and constant or cosine decay learning rate schedule. The architecture is dataset dependent: MLP for 2D Arcs, UNet (Ronneberger et al., 2015) for MNIST, and latent DiT (B/2) (Dosovitskiy et al., 2021; Rombach et al., 2022; Peebles and Xie, 2022) for ImageNet. For ImageNet, we use a pretrained VAE latent space. The FP-FM baselines all use a single neural network body with multiple output heads (Ingebrand et al., 2025). Thus, the cost of the basis functions in terms of parameter count is relatively minor. For example, on ImageNet, the FP-FM baselines use only $0.62\%$ more parameters than the conditional model. For all models, we integrate to $1 - \epsilon$ for $\epsilon = 1e - 3$ to avoid numerical instability.

We used 1,000 samples per task for Arc and 70 for MNIST. All FP-FM variants used 99 basis functions. Training batches contained 10 task distributions for Arc and 9 for MNIST, corresponding to 10,000 points and 630 images per optimization step, respectively. Models were trained for 10,000 gradient steps on Arc and 20,000 on MNIST. Samples were generated by integrating the learned vector field with a fixed-step fourth-order Runge–Kutta solver over 50 intervals.

All ImageNet models use the same DiT-B/2 backbone (Peebles and Xie, 2022), operating on $4\times32\times32$ latent representations of $256 \times 256$ images produced by a frozen Stable Diffusion VAE (sd-vae-ft-ema, latent scale 0.18215). The conditional baseline is the standard DiT-B/2 with class embeddings. FP-FM uses the same backbone but removes class conditioning entirely. Its only architectural modification is a wider output layer that predicts the marginal field $G_0$ together with $K$ basis fields, increasing the parameter count by less than $1\%$. We train all models from scratch with the flow-matching objective on 900 ImageNet classes. The remaining 100 classes are held out during training and are not seen by any method, including the baselines. At inference, each FP-FM variant is given

$m = 128$ context images from the target class. The corresponding coefficients are obtained by ridge-regularized least-squares projection onto the learned basis. This projection is performed once per class for Static FP-FM, at each time step for Temporal FP-FM, and at each state and time step for Dynamic FP-FM. We generate samples by integrating the resulting velocity field with a 50-step RK4 solver over $t \in [0, 1 - 10^{-3}]$. Classifier-free-style guidance is applied only to the class-dependent component of the field. For the fine-tuning baseline, we add rank-8 LoRA adapters to the attention and MLP projections of the conditional DiT and optimize them using the same 128 context images per class. For evaluation, each class provides 256 real images, divided into 128 context images and a disjoint set of 128 reference images. Every method generates 128 samples per class using the same set of 128 initial noise vectors. We compute FID, precision, and recall separately for each class using 128 generated and 128 reference images, and report the average over the 100 classes in each split.

Coefficients were obtained using ridge-regularized least squares, $c = (G^\top G + \lambda I)^{-1} G^\top f$. We used $\lambda = 10^{-3}$. In addition, we controlled basis scaling using $\|\mathrm{diag}(G^\top G) - 1\|_2^2$ as a regularizer during training (Ingebrand et al., 2025). This penalty encourages unit-norm basis functions and prevents the scale ambiguity of the least-squares objective from causing unbounded basis magnitudes.

2D Arcs and MNIST experiments are run on a desktop machine with a 3080. The ImageNet experiments are run on a cluster of 8 RTX PRO 6000 Blackwell GPUs. 2D Arcs experiments run in a few minutes, depending on the algorithm. MNIST experiments take a few hours. ImageNet experiments take about a day per algorithm.

For all approaches involving additional training, we train for 1000 gradient steps using the Adam optimizer with a learning rate of $1e-3$. For `Distribution-Guided`, we use the same architecture as the `Unconditional` model. For `Classifier-Guided`, we use the same architecture as the `Unconditional` model except slightly modified to output probabilities. We train the classifier using cross entropy with positive samples from $X_t^\iota$ and negative samples from $X_t^\kappa$ for $\kappa \neq \iota$. `Finetune` uses full finetuning for 2D Arcs and MNIST, and LoRA for ImageNet due to memory constraints.

ImageNet is significantly more challenging than the other datasets due to the high-dimensional and diverse nature of its images. We model each channel of the latent (which is a WxHxC image) as a separate function space, which slightly increases the expressivity of the coefficients. Note that $W \times H > k$. We also use one additional function encoder technique for ImageNet, which improves performance:

**Guided FP-FM** As introduced in (Ingebrand et al., 2025), the *residuals method* of function encoders represents a function as $f = \bar{f} + \sum_{i=1}^{k} c^i g^i$, where $\bar{f}$ is a neural net trained to model the average function in the training set. Therefore, the basis functions correct the error of mean function. We may leverage a similar technique in the flow matching setting, where $\bar{f}$ fits the mean velocity field, and the basis functions *guide* this velocity field towards a specific distribution, similar to classifier-free guidance.

## D Full Results

| Algorithm | Precision ↑ TD | UD | US | Recall ↑ TD | UD | US | Time (s) ↓ |
|---|---|---|---|---|---|---|---|
| `Unconditional` | 0.108 | 0.291 | 0.003 | 0.997 | 0.999 | 0.032 | 0.17 |
| | ± 0.013 | ± 0.036 | ± 0.001 | ± 0.002 | ± 0.001 | ± 0.005 | ± 0.01 |
| `Conditional` | 0.369 | 0.608 | 0.006 | 0.998 | 0.613 | 0.041 | 0.17 |
| | ± 0.049 | ± 0.045 | ± 0.001 | ± 0.002 | ± 0.026 | ± 0.006 | ± 0.01 |
| `Classifier-Guided` | 0.407 | 0.654 | 0.009 | 0.836 | 0.867 | 0.134 | 7.08 |
| | ± 0.040 | ± 0.081 | ± 0.001 | ± 0.016 | ± 0.019 | ± 0.068 | ± 0.22 |
| `Distribution-Guided` | 0.980 | 0.980 | 0.016 | 0.999 | 0.999 | 0.997 | 16.48 |
| | ± 0.002 | ± 0.003 | ± 0.002 | ± 0.000 | ± 0.000 | ± 0.002 | ± 0.28 |
| `Finetune` | 0.612 | 0.735 | 0.385 | 0.998 | 0.999 | 0.999 | 4.77 |
| | ± 0.033 | ± 0.018 | ± 0.019 | ± 0.001 | ± 0.001 | ± 0.001 | ± 0.12 |
| `Static FP-FM` | 0.324 | 0.320 | 0.086 | 0.847 | 0.811 | 0.894 | 0.21 |
| | ± 0.066 | ± 0.081 | ± 0.023 | ± 0.107 | ± 0.052 | ± 0.043 | ± 0.01 |
| `Temporal FP-FM` | 0.586 | 0.763 | 0.709 | 0.997 | 0.997 | 0.998 | 0.72 |
| | ± 0.043 | ± 0.025 | ± 0.012 | ± 0.002 | ± 0.001 | ± 0.001 | ± 0.00 |
| `Dynamic FP-FM` | 0.991 | 0.997 | 0.934 | 0.989 | 0.985 | 0.972 | 10.99 |
| | ± 0.001 | ± 0.001 | ± 0.002 | ± 0.001 | ± 0.002 | ± 0.002 | ± 0.04 |
| **Context** | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | - |
| | ± 0.001 | ± 0.000 | ± 0.001 | ± 0.000 | ± 0.001 | ± 0.000 | - |

Table 4: **2D Arc.** Comparison of methods across TD, UD, and US settings. Arrows indicate whether higher (↑) or lower (↓) is better. Results are averaged over 5 seeds. Standard deviation is shown after the ± symbol.

| Algorithm | Precision ↑ TD | UD | US | Recall ↑ TD | UD | US | FID ↓ TD | UD | US | Time (s) ↓ |
|---|---|---|---|---|---|---|---|---|---|---|
| `Unconditional` | 0.342 | 0.464 | 0.303 | 0.705 | 0.705 | 0.666 | 97.93 | 88.59 | 92.84 | 10.31 |
| | ± 0.172 | ± 0.233 | ± 0.165 | ± 0.353 | ± 0.353 | ± 0.337 | ± 123.96 | ± 126.66 | ± 134.69 | ± 0.29 |
| `Conditional` | 0.875 | 0.763 | 0.309 | 0.877 | 0.759 | 0.783 | 12.54 | 23.18 | 31.92 | 10.78 |
| | ± 0.011 | ± 0.077 | ± 0.059 | ± 0.013 | ± 0.048 | ± 0.074 | ± 0.91 | ± 5.10 | ± 3.19 | ± 0.19 |
| `Classifier-Guided` | 0.582 | 0.632 | 0.440 | 0.719 | 0.720 | 0.714 | 82.00 | 80.18 | 88.98 | 29.19 |
| | ± 0.292 | ± 0.317 | ± 0.227 | ± 0.360 | ± 0.360 | ± 0.360 | ± 131.12 | ± 127.91 | ± 137.06 | ± 0.32 |
| `Distribution-Guided` | 0.753 | 0.725 | 0.686 | 0.781 | 0.760 | 0.734 | 18.39 | 21.99 | 15.71 | 626.17 |
| | ± 0.040 | ± 0.051 | ± 0.119 | ± 0.073 | ± 0.157 | ± 0.117 | ± 1.44 | ± 4.35 | ± 2.00 | ± 2.19 |
| `Finetune` | 0.728 | 0.728 | 0.737 | 0.590 | 0.606 | 0.566 | 80.31 | 81.37 | 81.45 | 215.71 |
| | ± 0.364 | ± 0.364 | ± 0.370 | ± 0.297 | ± 0.303 | ± 0.285 | ± 132.90 | ± 129.52 | ± 139.73 | ± 0.16 |
| `Static FP-FM` | 0.783 | 0.757 | 0.554 | 0.865 | 0.856 | 0.766 | 16.71 | 21.03 | 22.53 | 10.63 |
| | ± 0.027 | ± 0.023 | ± 0.100 | ± 0.017 | ± 0.027 | ± 0.090 | ± 1.78 | ± 1.04 | ± 8.14 | ± 0.03 |
| `Temporal FP-FM` | 0.771 | 0.767 | 0.569 | 0.885 | 0.897 | 0.797 | 14.13 | 17.49 | 17.69 | 20.78 |
| | ± 0.028 | ± 0.024 | ± 0.115 | ± 0.018 | ± 0.014 | ± 0.029 | ± 0.52 | ± 0.37 | ± 1.64 | ± 0.01 |
| `Dynamic FP-FM` | 0.759 | 0.829 | 0.723 | 0.818 | 0.817 | 0.837 | 17.38 | 18.38 | 18.37 | 21.96 |
| | ± 0.081 | ± 0.061 | ± 0.114 | ± 0.032 | ± 0.022 | ± 0.078 | ± 4.03 | ± 3.22 | ± 7.02 | ± 0.01 |
| **Context** | 0.891 | 0.898 | 0.900 | 0.897 | 0.893 | 0.903 | 11.16 | 12.91 | 10.27 | - |
| | ± 0.015 | ± 0.012 | ± 0.066 | ± 0.009 | ± 0.010 | ± 0.019 | ± 0.35 | ± 0.37 | ± 0.32 | - |

Table 5: **MNIST.** Comparison of methods across TD, UD, and US settings. Arrows indicate whether higher (↑) or lower (↓) is better. Results are averaged over 5 seeds. Standard deviation is shown after the ± symbol.

# E Ablations

## E.1 Number of Shots

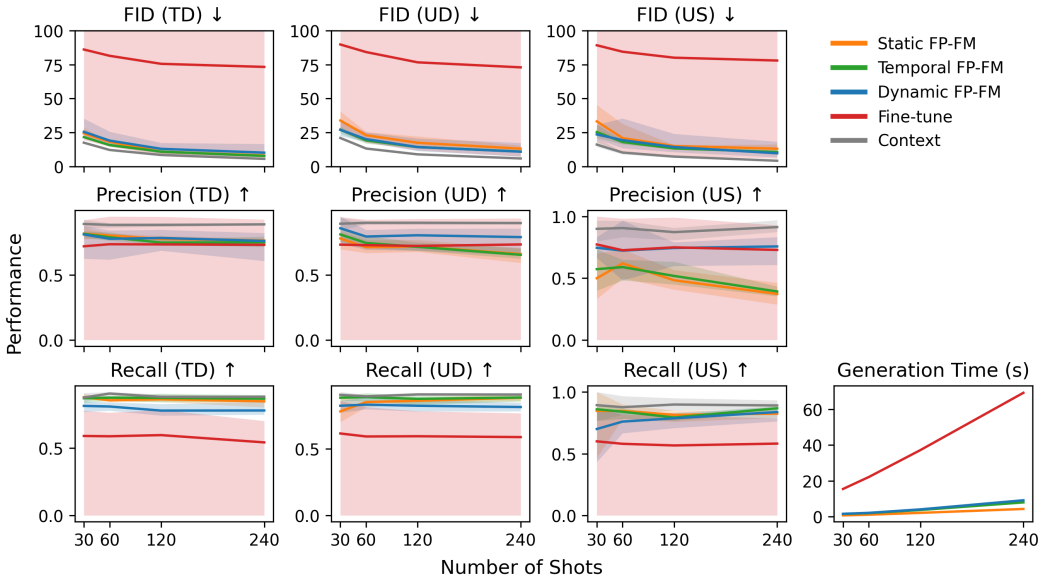


Figure 6: **Ablation on the Number of Shots.** We vary the number of shots both during training and evaluation for all three FP-FM baselines over 5 seeds. Plots show the mean, min, and max for each algorithm.

We vary the number of shots provided during training and evaluation on the MNIST dataset. We observe that for all three FP-FM variants, FID decreases as the number of shots increases. `Dynamic FP-FM` has stable precision and recall as the number of shots increases. For `Static FP-FM` and `Temporal FP-FM`, precision may slightly decrease as the number of shots increases. **Context** provides an approximate upper bound on performance as it compares the ground truth provided context to the real images.

## E.2 Number of Basis Functions

We likewise ablate the number of basis functions on the MNIST dataset. We find that none of `Static FP-FM`, `Temporal FP-FM`, or `Dynamic FP-FM` are very sensitive to the number of basis functions on this dataset after a certain threshold is reached. In other words, adding more basis functions does not degrade performance. Generation time slightly increases as the number of basis functions increases because the cost of the least-squares solve increases. Since `Dynamic FP-FM` requires the most least-squares solves, its generation time is the most affected of all three FP-FM variants, though the increase is relatively small.

## E.3 Dynamic FP-FM Overfitting Analysis

Dynamic FP-FM is susceptible to overfitting due to the importance sampling estimator. However, the basis functions play a crucial role in preventing this overfitting. To verify this, we introduce a nearest-neighbor statistic $\tau$: the median distance from a generated sample to the calibration set, divided by the median distance from a held-out real image of the same class to that same set. A value of $\tau = 1$ means generations sit exactly as far from the provided examples as a fresh photograph does; $\tau << 1$ indicates copying. We also report the fraction of generations falling below the 5th percentile of the real-image distances, for which 0.05 is expected by construction.

We find that the nearest neighbor statistic $\tau = 1.035$, and the fraction of generations falling below the 5th percentile of real distances is 0.075. So, Dynamic FP-FM is only slightly overfitting.

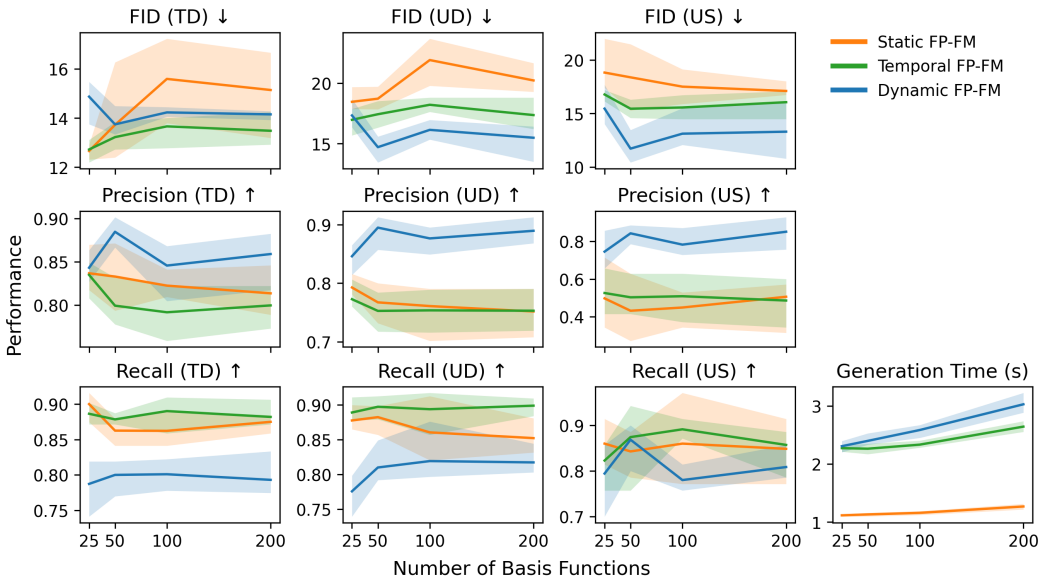


Figure 7: **Ablation on the Number of Basis Functions.** We vary the number of basis functions for all three FP-FM baselines on the MNIST dataset.

## F Extended Discussion and FAQ

The questions below address the main technical and experimental concerns that might arise when interpreting our results. We explain when the basis projection is meaningful, why it regularizes the empirical velocity estimate, and why its added generation cost is small. We also clarify the parameter comparison, baseline choices, and number of basis functions, while stating the limits of the method in few-shot settings.

**Why is Equation (11) not singular and/or trivial?** Equation (11) projects the $n$-dimensional estimated velocity onto a set of $k$ $n$-dimensional basis functions. As argued in prior work (Ingebrand et al., 2025), the learned basis functions are independent at initialization, and the training loss discourages dependence, so it is typically the case that the basis is independent. Therefore, we can view this as finding the nearest vector within the $k$-dimensional subspace of $\mathbb{R}^n$. When $k < n$, this projection indeed regularizes the estimator by forcing the result to lie within the span of the basis. For most image-based problems, $k < n$. If $k \geq n$, then the projection becomes trivial as the basis spans all of $\mathbb{R}^n$. In particular, this occurs on the toy example 2D Arcs.

**Does this work in few-shot settings?** Specifying a data manifold for high-dimensional manifolds such as images from only a few data points is ill-posed. So, the method is still applicable, but we expect the results to be quite poor because there simply is not enough information to estimate the manifold. We believe no method that relies on example data alone should perform well in such a setting without making heavy use of priors on what the distributions should look like. Incorporating such priors to target the few-shot setting is beyond the scope of this work.

**Isn't the computational complexity of `Dynamic FP-FM` quite high during generation?** The main expense for generating data is forward passing the neural network. Any data processing, such as computing the empirical velocity estimate, is extremely cheap. Both `Dynamic FP-FM` and other methods such as `Finetune` require one model forward pass per generation step. `Dynamic FP-FM` additionally requires data processing and a $k$-dimensional least squares calculation. Both of these are negligible in comparison to the forward pass, and hence the marginal cost of generating data is only slightly higher than `Conditional` or `Finetune`.

**Weight collapse in Theorem 1?** In high-dimensional data spaces $\mathbb{R}^n$, the weights assigned to each sample tend to concentrate around a single sample. The basis functions play the crucial role of preventing generation from always following a single sample. Because we project this vector onto the basis, the output cannot directly overfit to empirical estimate, and the velocity field instead points in a direction that is generally useful for generation as prescribed by the learned basis. Consequently, one interpretation of `Dynamic FP-FM` is a learned, regularized version of a purely data-driven empirical estimate.

**Why does the `Conditional` baseline use a one-hot embedding?** One-hot embeddings are the standard approach for conditioning on a discrete set of values, in this case the target class. One may wonder why we do not use an embedding table, similar to a language model. In fact, these two methods are mathematically equivalent. Consider multiplying a matrix $W$ by the one-hot vector $v$ with $v_i = 1$ and $v_j = 0$ for $j \neq i$. The output will be the $i$'th column of $W$, which is exactly the same as simply "looking up" the $i$'th column using index $i$ rather than a vector $v$. Therefore, embedding tables are effectively sparse implementations of one-hot embeddings.

**Doesn't FP-FM require $k$-times as many parameters as a `Conditional` model? Is the difference in performance due to this extra capacity?** As is typical for function encoders (Ingebrand et al., 2025), we use a single body with multiple output heads to parameterize the basis. Effectively, this means FP-FM shares almost entirely the same architecture as `Conditional`, except in the last few layers. Therefore, the difference in parameters is relatively small and does not explain the difference in performance. In particular, the total number of parameters is only 0.62% larger for `Dynamic FP-FM` relative to `Conditional` on ImageNet.

**How do you choose the number of basis functions?** As shown in prior work (Ingebrand et al., 2025), using too many basis functions does not negatively impact performance. Therefore, it is best to err on the side of caution and use more basis functions than needed. To that end, we use 99 basis functions for 2D Arcs and MNIST, and 512 for ImageNet. Prior work has also investigated how to determine the minimum number of basis functions exactly, see Low et al. (2025).

**`Conditional` obviously won't generalize to unseen classes. Why aren't there context-conditioned baselines?** This is exactly the argument we are trying to make. Any model trained on conditioning variables will only work for distributions representable within that conditioning variable. Therefore, these models rely heavily on diverse training in order to generalize to related but unseen context variables. This limitation is a major motivation for our work and why we seek a context-conditioned model. Note that we aren't suggesting there is no place for conditional models. They are a reasonable approach in many settings. However, our results show that a context-conditioned model can perform comparatively, and excels in some areas where conditional models fail. In terms of other context-conditioned baselines, we use `Classifier-Guided`, `Distribution-Guided`, and `Finetune` as representative approaches. One other obvious choice is to directly condition a transformer on the context, which consists of 10's of images. However, under a standard image processor such as Pan and Scan (Team et al., 2025), each image is converted into hundreds of tokens, leading to a joint context space of tens of thousands of tokens, not counting the generated image itself. While this is mathematically reasonable, the quadratic cost of attention implies it far exceeds all other approaches in expense. Thus, it is beyond the scope of this work.

| | Precision ↑ | | | Recall ↑ | | | FID ↓ | | | Time (s) ↓ |
|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | TD | UD | US | TD | UD | US | TD | UD | US | |
| `Leave-One-Out` | 0.792 | 0.791 | 0.734 | 0.640 | 0.702 | 0.637 | 18.01 | 20.03 | 17.85 | 21.58 |
| | ± 0.017 | ± 0.024 | ± 0.084 | ± 0.012 | ± 0.038 | ± 0.106 | ± 1.03 | ± 1.31 | ± 1.61 | ± 0.06 |
| `Dynamic FP-FM` | 0.759 | 0.829 | 0.723 | 0.818 | 0.817 | 0.837 | 17.38 | 18.38 | 18.37 | 21.96 |
| | ± 0.081 | ± 0.061 | ± 0.114 | ± 0.032 | ± 0.022 | ± 0.078 | ± 4.03 | ± 3.22 | ± 7.02 | ± 0.01 |

Table 6: **Leave-one-out ablation on MNIST.** Comparison of `Leave-One-Out` relative to `Dynamic FP-FM` across TD, UD, and US settings. Arrows indicate whether higher (↑) or lower (↓) is better. Results are averaged over 5 seeds. Standard deviation is shown after the ± symbol.

**Why are the basis functions independent?** As argued in prior work (Ingebrand et al., 2025), the basis functions are almost surely independent upon initialization. The argument is based on sampling $k$ vectors from an infinite dimensional vector space. During training, dependent basis functions have higher loss, and hence the basis functions to not converge towards dependence. Thus, we find that the basis is independent in practice.

**Is it an issue that Algorithm 3 uses the same data to generate $x_t^{\iota,i}$ as to compute the empirical velocity estimate?** We run an ablation where we prevent the $x_0^{\iota,i}$ that was used to generate $x_t^{\iota,i}$ from affecting the empirical estimate. To do so, we set its log probability to negative infinity before normalizing, guaranteeing its probability is set to 0. We call this variant `Leave-One-Out`. We find that it has slightly worse recall, but otherwise similar performance as `Dynamic FP-FM`. So, this detail is not of much practical relevance. See Table 6.

## G ALGORITHMS

In this section, we provide the algorithm for each of our methods. Note that they largely follow the algorithm in (Ingebrand et al., 2025). The main difference between them is the sampling procedure. Additionally, function encoders typically use different data sets for calibrating the coefficients and computing the loss. However, this requires forward passing both datasets, which is expensive for image generation. Instead we use the same data for both calibration and training loss, which improves computational efficiency.

**Algorithm 1** `Static FP-FM`

**given** datasets $\{\mathcal{D}^\iota\}_{\iota\in\mathcal{T}}$, learning rate $\alpha$
Initialize basis $\{g_1, \dots, g_k\}$ with parameters $\theta$
**while** not converged **do**
  *// Initialize loss for gradient accumulation.*
  $L \leftarrow 0$
  *// For every distribution in the training set.*
  **for all** $\iota \in \mathcal{T}$ **do**
    *// Define the samples.*
    $\{x_1^{\iota,i}\}_{i=1}^m \leftarrow \mathcal{D}^\iota$
    $\{x_0^{\iota,i}\}_{i=1}^m \sim \mathcal{N}(0, I)$
    $\{t^i\}_{i=1}^m \sim \text{Unif}([0,1])$
    *// Standard $X_t$ definition.*
    $x_t^{\iota,i} \leftarrow (1-t^i)x_0^{\iota,i} + t^i x_1^{\iota,i}$
    *// Compute coefficients. The inner product is approximated using the samples.*

$$c^\iota \leftarrow \left( \begin{bmatrix} \langle g^1, g^1\rangle_{p_{t,X_t^\iota}} & \cdots & \langle g^1, g^k\rangle_{p_{t,X_t^\iota}} \\ \vdots & \ddots & \vdots \\ \langle g^k, g^1\rangle_{p_{t,X_t^\iota}} & \cdots & \langle g^k, g^k\rangle_{p_{t,X_t^\iota}} \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \langle v^\iota, g^1\rangle_{p_{t,X_t^\iota}} \\ \vdots \\ \langle v^\iota, g^k\rangle_{p_{t,X_t^\iota}} \end{bmatrix}$$

    *// Define the approximated velocity field.*
    $\hat{v}^\iota(\cdot,\cdot) := \sum_{i=1}^k c^{\iota,i} g^i(\cdot,\cdot)$
    *// Gradient accumulation. The norm is again approximated using the samples.*
    $L \leftarrow L + \|v^\iota - \hat{v}^\iota\|^2_{p_{t,X_t^\iota}}$
  **end for**
  *// Gradient descent.*
  $\theta \leftarrow \theta - \alpha \nabla_\theta(L)$
**end while**
**return** $\{g_1, \dots, g_k\}$

**Algorithm 2** `Temporal FP-FM`

**given** datasets $\{\mathcal{D}^\iota\}_{\iota\in\mathcal{T}}$, learning rate $\alpha$
Initialize basis $\{g_1, \dots, g_k\}$ with parameters $\theta$
**while** not converged **do**
  *// Initialize loss for gradient accumulation.*
  $L \leftarrow 0$
  *// For every distribution in the training set.*
  **for all** $\iota \in \mathcal{T}$ **do**
    *// Important: Sample a single time t*
    $t \sim \text{Unif}([0,1])$
    *// Define the samples.*
    $\{x_1^{\iota,i}\}_{i=1}^m \leftarrow \mathcal{D}^\iota$
    $\{x_0^{\iota,i}\}_{i=1}^m \sim \mathcal{N}(0, I)$
    *// Standard $X_t$ definition.*
    $x_t^{\iota,i} \leftarrow (1-t)x_0^{\iota,i} + t x_1^{\iota,i}$
    *// Compute coefficients. The inner product is approximated using the samples.*

$$c^\iota(t) \leftarrow \left( \begin{bmatrix} \langle g_t^1, g_t^1 \rangle_{p_{X_t^\iota|t}} & \cdots & \langle g_t^1, g_t^k \rangle_{p_{X_t^\iota|t}} \\ \vdots & \ddots & \vdots \\ \langle g_t^k, g_t^1 \rangle_{p_{X_t^\iota|t}} & \cdots & \langle g_t^k, g_t^k \rangle_{p_{X_t^\iota|t}} \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \langle v^\iota, g_t^1 \rangle_{p_{X_t^\iota|t}} \\ \vdots \\ \langle v^\iota, g_t^k \rangle_{p_{X_t^\iota|t}} \end{bmatrix}$$

    *// Define the approximated velocity field.*
    $\hat{v}_t^\iota(\cdot) := \sum_{i=1}^k c^{\iota,i}(t) g_t^i(\cdot)$
    *// Gradient accumulation. The norm is again approximated using the samples.*
    $L \leftarrow L + \|v_t^\iota - \hat{v}_t^\iota\|^2_{p_{X_t^\iota|t}}$
  **end for**
  *// Gradient descent.*
  $\theta \leftarrow \theta - \alpha \nabla_\theta(L)$
**end while**
**return** $\{g_1, \dots, g_k\}$

**Algorithm 3** `Dynamic FP-FM`

**given** datasets $\{\mathcal{D}^\iota\}_{\iota\in\mathcal{T}}$, learning rate $\alpha$
Initialize basis $\{g_1,\dots,g_k\}$ with parameters $\theta$
**while** not converged **do**
  *// Initialize loss for gradient accumulation.*
  $L \leftarrow 0$
  *// For every distribution in the training set.*
  **for all** $\iota \in \mathcal{T}$ **do**
    *// Define the samples.*
    $\{x_1^{\iota,i}\}_{i=1}^m \leftarrow \mathcal{D}^\iota$
    $\{x_0^{\iota,i}\}_{i=1}^m \sim \mathcal{N}(0,I)$
    $\{t^i\}_{i=1}^m \sim \mathrm{Unif}([0,1])$
    *// Standard $X_t$ definition.*
    $x_t^{\iota,i} \leftarrow (1-t^i)x_0^{\iota,i} + t^i x_1^{\iota,i}$
    *// Important: In the following for loop, we compute the noise samples from $x_t$ and $x_1$.*
    *// Therefore we delete the current noise samples to make this explicit.*
    Delete $\{x_0^{\iota,i}\}_{i=1}^m$
    *// For every state-time.*
    **for all** $(x_t,t) \in \{(x_t^{\iota,j}, t^j)\}_{j=1}^m$ **do**
      *// Compute $x_0^*$ as in Theorem 1*
      *// For each $x_t$, compute $x_0^{\iota,i}$ for all $i$, meaning we have a set $\{x_0^{\iota,i}\}_{i=1}^m$*
      $x_0^{\iota,i} \leftarrow \frac{x_t - t x_1^{\iota,i}}{1-t}$
      *// Compute coefficients. The inner product is approximated using the samples.*
      *// See Appendix A.*

$$c^\iota(x_t,t) \leftarrow \left( \begin{bmatrix} \langle g^1, g^1\rangle_{p_{t,X_t^\iota|t,X_t}} & \cdots & \langle g^1, g^k\rangle_{p_{t,X_t^\iota|t,X_t}} \\ \vdots & \ddots & \vdots \\ \langle g^k, g^1\rangle_{p_{t,X_t^\iota|t,X_t}} & \cdots & \langle g^k, g^k\rangle_{p_{t,X_t^\iota|t,X_t}} \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \langle v^\iota, g^1\rangle_{p_{t,X_t^\iota|t,X_t}} \\ \vdots \\ \langle v^\iota, g^k\rangle_{p_{t,X_t^\iota|t,X_t}} \end{bmatrix}$$

      *// Define the approximated velocity field.*
      $\hat{v}^\iota(x_t,t) := \sum_{i=1}^k c^{\iota,i}(x_t,t) g^i(x_t,t)$
      *// Gradient accumulation. The norm is again approximated using the samples.*
      $L \leftarrow L + \|v^\iota - \hat{v}^\iota\|^2_{p_{t,X_t^\iota|t,X_t}}$
    **end for**
  **end for**
  *// Gradient descent.*
  $\theta \leftarrow \theta - \alpha \nabla_\theta(L)$
**end while**
**return** $\{g_1,\dots,g_k\}$