PROTOFLOW: PROTOTYPE-GUIDED FLOW MATCHING FOR MULTIVARIATE TIME SERIES FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative modeling has shown strong promise for multivariate time mseries (MTS) forecasting, especially scale to high-dimensional settings. Diffusion-based methods achieve competitive performance but typically require many sampling steps at inference. VAE-based non-iterative forecasting frameworks have therefore emerged as an efficient alternative. Within this line of work, vector quantization (VQ) enables controllable latent space modeling by mapping multivariate series into compact discrete representations. Existing VQ-based forecasting methods, however, typically rely on autoregressive (AR) token generation, which suffers from exposure bias and training-inference mismatch. Flow matching provides an efficient non-autoregressive alternative for latent forecasting, but existing formulations usually initialize transport from a generic Gaussian prior. We instead observe that the trained VQ codebook already captures representative latent prototypes and can thus serve as a more informative prior for flow matching. Based on this insight, we propose **ProtoFlow**, a forecasting framework that combines vector-quantized autoencoding with **Proto**type-prior **Flow** matching. Our method first maps multivariate sequences into a discrete latent space, then constructs a structured prior from the learned codebook, and finally learns a DiT-based rectified flow to transport samples from this prior to future latent representations conditioned on historical observations. By replacing generic noise initialization with a learned prototype prior, ProtoFlow avoids the rollout mismatch of AR token prediction and promotes faster training convergence. Extensive experiments on benchmark datasets show that it consistently achieves superior forecasting performance with efficient inference.

1 INTRODUCTION

Multivariate time series (MTS) forecasting is an important research problem with broad applications in domains such as economics Huang et al. (2024), traffic systems Shao et al. (2022); Jiang et al. (2023), energy management Chen et al. (2024), and weather prediction Wu et al. (2020; 2023). In many real-world scenarios, forecasting involves a large number of variables evolving over time, making high-dimensional MTS forecasting particularly challenging. As a result, developing effective forecasting models for high-dimensional multivariate sequences has become an increasingly important topic.

Generative models provide a natural and effective framework for this setting, and have shown strong potential for scaling to high-dimensional MTS forecasting Sohl-Dickstein et al. (2015); Dinh et al. (2016); Kingma et al. (2019); Song et al. (2020). Among them, diffusion-based methods have achieved competitive performance in recent years Rasul et al. (2021); Shen & Kwok (2023); Meijer & Chen (2024); Cao et al. (2024); Wang et al. (2024); Ruan et al. (2025). However, their inference typically requires many denoising steps, leading to substantial computational cost.

This limitation has motivated increasing interest in VAEs-based non-iterative forecasting frameworks, which perform generation in a learned latent space. Within this family, vector quantization (VQ) provides a controllable latent space modeling strategy by mapping complex multivariate sequences into a compact discrete codebook. Such vector-quantized representations have recently shown promise in token-based time series modeling and forecasting Talukder et al. (2024); Chen et al. (2024); Feng et al. (2025). However, existing VQ-based forecasting methods typically rely on

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.

autoregressive (AR) token generation. Since AR models are usually trained with teacher forcing, they depend on ground-truth previous tokens during training but on their own predictions during inference, resulting in train–test mismatch and error accumulation during rollout Schmidt (2019); Bengio et al. (2015).

Flow matching (FM) learns continuous transport and can support non-autoregressive generation Liu et al. (2022). Recent time series applications include TSFlow, which introduces Gaussian-process priors Kollovieh et al. (2024); TFM, which learns history-conditioned Neural SDEs through simulation-free training Zhang et al. (2024); and FlowTime, which adopts teacher-forced autoregressive forecasting El-Gazzar & van Gerven (2025). These methods primarily model transport in observation space, where high-dimensional MTS forecasting requires capturing complex temporal and cross-variable dependencies in the joint predictive distribution. ProtoFlow addresses this challenge by jointly generating future sequences in a compact vector-quantized latent space. Its prototype prior incorporates learned latent centers, empirical frequencies, and local variability into the source distribution, providing a structured initialization for conditional transport. Empirically, this initialization supports earlier FM training stabilization than its Gaussian-prior counterpart, while enabling accurate few-step forecasting in high-dimensional MTS settings.

Based on this observation, we propose **ProtoFlow**, a prototype-prior flow matching framework for high-dimensional multivariate time series forecasting. ProtoFlow combines vector-quantized representations with a structured source prior derived from learned codebook prototypes and their empirical statistics. It jointly generates future latent sequences through non-autoregressive flow matching, avoiding the ground-truth versus generated-token conditioning mismatch associated with teacher-forced autoregressive rollout. Experiments on five real-world benchmarks with several prediction horizons demonstrate strong overall performance against competitive diffusion- and flow-based baselines with few ODE sampling steps. Further analyses show earlier FM training stabilization and reduced sampling-step sensitivity relative to Gaussian initialization, supporting the effectiveness of reusing learned latent structure for transport initialization.

Main Contributions.

- We propose **ProtoFlow**, a framework for high-dimensional multivariate time series forecasting that combines vector-quantized representations with non-autoregressive flow matching to jointly generate future latent sequences, avoiding the conditioning mismatch associated with teacher-forced autoregressive rollout.
- We develop a structured source prior from learned prototype centers and empirical frequencies, extending the codebook's role from latent quantization to transport initialization. Compared with Gaussian initialization, this design empirically supports earlier FM training stabilization and reduced sensitivity to the sampling-step budget.
- Experiments on five real-world benchmarks with testing prediction horizons demonstrate strong overall probabilistic and deterministic forecasting performance against competitive diffusion- and flow-based baselines with substantially fewer sampling steps.

2 PRELIMINARIES

We briefly review vector-quantized variational autoencoders (VQ-VAE) Van Den Oord et al. (2017) and rectified flow matching, which underpin ProtoFlow.

VQ-VAE Modeling. A VQ-VAE comprises an encoder $\mathcal{E}_\psi$, a decoder $\mathcal{G}_\phi$, and a learned codebook $\mathcal{Z} \in \mathbb{R}^{K \times n_z}$. Given a time series $\mathcal{X}_t \in \mathbb{R}^{\tau \times D}$, the encoder produces $\hat{z} = \mathcal{E}_\psi(\mathcal{X}_t) \in \mathbb{R}^{L \times n_z}$, where $L = \lfloor \tau/\rho \rfloor$ and ρ is the temporal downsampling ratio. Standard vector quantization assigns each latent vector to its nearest codebook entry:

$$a_\ell = \arg\min_{k \in \{1,\dots,K\}} \|\hat{z}^\ell - \mathcal{Z}_k\|_2^2, \qquad z_q^\ell = \mathcal{Z}_{a_\ell}. \tag{1}$$

The discrete index sequence $\mathbf{a} = (a_1, \dots, a_L)$ identifies the quantized embeddings $z_q = q(\hat{z})$, from which the decoder reconstructs $\hat{\mathcal{X}}_t = \mathcal{G}_\phi(z_q)$.

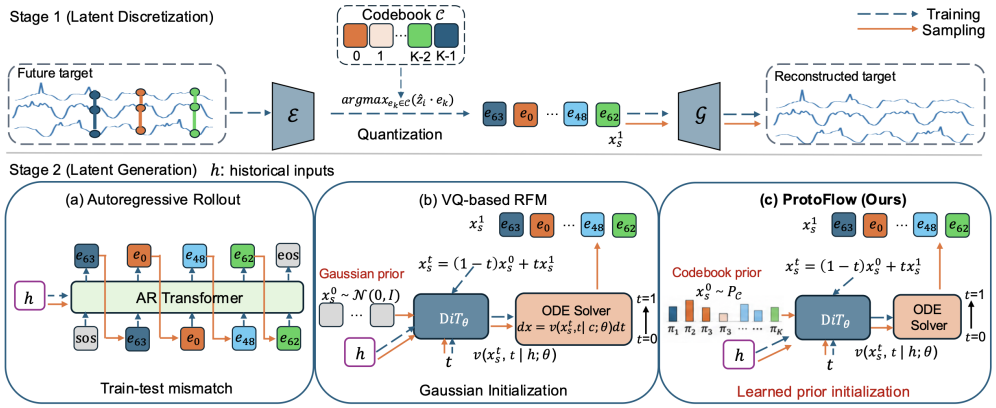

Figure 1: Illustration of our ProtoFlow. Stage 1 discretizes the future target sequence into compact latent tokens via similarity-based VQ latent modeling. Stage 2 compares three latent-generation paradigms under the same tokenized space: (a) Autoregressive rollout, which suffers from training–inference mismatch; (b) Gaussian-initialized flow matching, which starts transport from an unstructured prior; and (c) The proposed ProtoFlow, which initializes conditional rectified flow from a learned codebook prior for latent flow matching generation. The future time features y^c is omitted here.

Autoregressive token-based forecasting models generate these indices conditioned on historical information $\mathbf{c}$:

$$p(\mathbf{a} \mid \mathbf{c}) = \prod_{\ell=1}^{L} p(a_\ell \mid a_{<\ell}, \mathbf{c}). \tag{2}$$

Under teacher-forced training, conditioning uses ground-truth preceding tokens, whereas inference relies on generated tokens. This discrepancy can propagate prediction errors during sequential generation, motivating non-autoregressive alternatives for latent forecasting.

Rectified Flow Matching (RFM) Liu et al. (2022). RFM learns a velocity field that transports samples from a source distribution to a target distribution. Given conditioning information $\boldsymbol{c}$, independently sample $\boldsymbol{x}_0 \sim \pi_0$ and $\boldsymbol{x}_1 \sim \pi_1(\cdot \mid \boldsymbol{c})$, and construct

$$\boldsymbol{x}_t = (1-t)\boldsymbol{x}_0 + t\boldsymbol{x}_1, \qquad t \sim \mathcal{U}(0,1). \tag{3}$$

The network $\boldsymbol{v}_\theta$ is trained to predict the displacement $\boldsymbol{x}_1 - \boldsymbol{x}_0$:

$$\mathcal{L}_{\mathrm{RFM}}(\theta) = \mathbb{E}_{\boldsymbol{c},t,\boldsymbol{x}_0,\boldsymbol{x}_1}\left[\|\boldsymbol{v}_\theta(\boldsymbol{x}_t, t \mid \boldsymbol{c}) - (\boldsymbol{x}_1 - \boldsymbol{x}_0)\|_2^2\right]. \tag{4}$$

At inference, a source sample is transformed by numerically integrating the learned ODE:

$$\frac{\mathrm{d}\hat{\boldsymbol{x}}_t}{\mathrm{d}t} = \boldsymbol{v}_\theta(\hat{\boldsymbol{x}}_t, t \mid \boldsymbol{c}), \qquad \hat{\boldsymbol{x}}_0 \sim \pi_0. \tag{5}$$

ProtoFlow applies this framework to vector-quantized latent embeddings, using a structured source prior derived from learned codebook prototypes and their training statistics to initialize conditional transport.

3 Method

In this section, we present **ProtoFlow**, a prototype-prior flow-matching framework for multivariate time series forecasting. ProtoFlow first learns a vector-quantized tokenizer to map future sequences into a compact discrete latent space, and then performs conditional rectified flow matching initialized from a learned codebook prior. Thus, the codebook serves not only as a latent discretizer, but also as a structured prior for latent generation.

3.1 Vector-Quantized Tokenizer

Stage 1 learns a vector-quantized tokenizer that maps future target sequences into compact latent representations for subsequent flow-based forecasting. Given a target sequence $\mathcal{X}_t \in \mathbb{R}^{\tau \times D}$, we use

an encoder $\mathcal{E}_\psi$, a decoder $\mathcal{G}_\phi$, and a learned codebook $\mathcal{Z} \in \mathbb{R}^{K \times n_z}$ containing K embeddings of dimension n_z. The encoder produces a latent sequence $\hat{z} = \mathcal{E}_\psi(\mathcal{X}_t) \in \mathbb{R}^{L \times n_z}$, where $L = \lfloor \tau/\rho \rfloor$ and ρ denotes the temporal downsampling ratio.

Following SDformer Chen et al. (2024), we adopt similarity-driven vector quantization. Specifically, we ℓ_2-normalize the encoder outputs and codebook embeddings, and assign each latent vector $\hat{z}^\ell$ to the code with the highest cosine similarity:

$$a_\ell = \underset{k \in \{1,\dots,K\}}{\arg\max} \frac{\langle \hat{z}^\ell, \mathcal{Z}_k \rangle}{\|\hat{z}^\ell\|_2 \|\mathcal{Z}_k\|_2}, \qquad z_q^\ell = \mathcal{Z}_{a_\ell}. \tag{6}$$

For unit-normalized vectors, this criterion reduces to maximizing their inner product, making code assignment depend on latent direction rather than magnitude. The resulting quantized embedding sequence is

$$\mathcal{X}_s = [z_q^1, \dots, z_q^L] = q(\mathcal{E}_\psi(\mathcal{X}_t)), \tag{7}$$

which provides the latent target representation for Stage 2.

The decoder reconstructs the target sequence as $\hat{\mathcal{X}}_t = \mathcal{G}_\phi(\mathcal{X}_s)$. We train the tokenizer using a reconstruction loss and a cosine-based commitment loss:

$$\mathcal{L}_{\mathrm{VQ}} = \|\mathcal{X}_t - \hat{\mathcal{X}}_t\|_2^2 + \frac{\beta}{L} \sum_{\ell=1}^{L} \left(1 - \left\langle \hat{z}^\ell, \mathrm{sg}[z_q^\ell] \right\rangle\right), \tag{8}$$

where $z_q = \mathcal{X}_s$, $\hat{z}^\ell$ and z_q^ℓ are ℓ_2-normalized encoder and codebook embeddings, respectively, and $\mathrm{sg}[\cdot]$ denotes the stop-gradient operator. The coefficient β controls the commitment strength, encouraging encoder outputs to align with their assigned codebook embeddings. We update the codebook using exponential moving averages and reset inactive entries to stabilize codebook learning, which is shown in Appendix Eqn 39.

3.2 Codebook-Prior Rectified Flow Matching

After learning the Stage 1 tokenizer, we model the conditional distribution of future latent sequences using rectified flow matching. The target embedding sequence is $\mathbf{x}_s^1 = \mathcal{X}_s \in \mathbb{R}^{L \times n_z}$, where $L = \lfloor \tau/\rho \rfloor$ and ρ denotes the temporal downsampling ratio. The superscripts 0 and 1 denote the source and target endpoints of latent transport, respectively.

Codebook-prior Initialization. A key design of ProtoFlow is to initialize latent transport from a structured codebook prior. We construct this prior from the learned codebook $\mathcal{Z}$ and training latent statistics. Each component is specified by a mixing weight π_k, a prototype center $\mu_k \in \mathbb{R}^{n_z}$ associated with the k-th codebook entry, and local perturbation parameters (B_k, σ_k). Here, $B_k \in \mathbb{R}^{n_z \times r}$ contains the principal local basis directions, $\sigma_k \in \mathbb{R}^r_{\geq 0}$ contains their scales, and r denotes the local covariance rank. The single-position source prior is

$$p_{\mathcal{Z}}(z) = \sum_{k=1}^{K} \pi_k \, p_k(z; \mu_k, B_k, \sigma_k, \gamma), \qquad \pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1, \tag{9}$$

where $\gamma \geq 0$ controls the perturbation magnitude, and each component is defined by the normalized sampling procedure below.

For each latent position ℓ, we independently sample a component index and a local perturbation:

$$s_\ell^0 \sim \mathrm{Cat}(\pi_1, \dots, \pi_K), \qquad \varepsilon_\ell \sim \mathcal{N}(0, I_r), \tag{10}$$

and construct

$$\mathbf{x}_{s,\ell}^0 = \mathrm{Norm}\Big(\mu_{s_\ell^0} + \gamma B_{s_\ell^0}\big(\sigma_{s_\ell^0} \odot \varepsilon_\ell\big)\Big), \tag{11}$$

where $\mathrm{Norm}(\cdot)$ denotes ℓ_2 normalization and $\odot$ denotes elementwise multiplication. The parameters (B_k, σ_k) describe the local perturbation before normalization. Stacking all positions yields the source latent sequence

$$\mathbf{x}_s^0 = [x_{s,1}^0, x_{s,2}^0, \dots, x_{s,L}^0]. \tag{12}$$

This initialization is therefore centered on learned latent prototypes and their local neighborhoods, rather than on a generic isotropic distribution.

Conditional Rectified Flow Matching. Let $\mathbf{c} = (\mathcal{H}_p, \mathbf{y}^c)$ denote the forecasting conditions, comprising the encoded historical context and known temporal features for the prediction window. For a target–condition pair $(\mathbf{x}_s^1, \mathbf{c})$ and an independently sampled source $\mathbf{x}_s^0 \sim P_{\mathcal{Z}}$, we define the linear interpolation

$$\mathbf{x}_s^t = (1-t)\mathbf{x}_s^0 + t\mathbf{x}_s^1, \qquad t \in [0,1], \tag{13}$$

where t denotes flow time. Its target velocity is

$$\mathbf{u}_s^t = \frac{\mathrm{d}\mathbf{x}_s^t}{\mathrm{d}t} = \mathbf{x}_s^1 - \mathbf{x}_s^0. \tag{14}$$

We parameterize the vector field as $\mathbf{v}_\theta(\mathbf{x}_s^t, t \mid \mathbf{c}) = \mathrm{DiT}_\theta(\mathbf{x}_s^t, t, \mathbf{c})$ and minimize

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{\substack{(\mathbf{x}_s^1, \mathbf{c}) \sim q_{\mathrm{data}}, \\ \mathbf{x}_s^0 \sim P_{\mathcal{Z}},\ t \sim q_t}} \left[\left\| \mathbf{v}_\theta(\mathbf{x}_s^t, t \mid \mathbf{c}) - (\mathbf{x}_s^1 - \mathbf{x}_s^0) \right\|_F^2 \right], \tag{15}$$

where q_{data} denotes the empirical distribution of target latent sequences and their conditions, q_t is the training-time sampling distribution on $[0,1]$, and $\|\cdot\|_F$ denotes the Frobenius norm.

Sampling. At inference, we sample $\hat{\mathbf{x}}_s^{t_0} \sim P_{\mathcal{Z}}$ and integrate the learned vector field over a grid $0 = t_0 < t_1 < \cdots < t_M = 1$ using Euler updates:

$$\hat{\mathbf{x}}_s^{t_{j+1}} = \hat{\mathbf{x}}_s^{t_j} + \Delta t_j \, \mathbf{v}_\theta\big(\hat{\mathbf{x}}_s^{t_j}, t_j \mid \mathbf{c}\big), \qquad \Delta t_j = t_{j+1} - t_j, \tag{16}$$

for $j = 0, \ldots, M-1$. Each update jointly evolves the entire future latent sequence, enabling non-autoregressive forecast generation. The final latent sample is decoded into the forecasting target:

$$\hat{\mathcal{X}}_t = \mathcal{G}_\phi(\hat{\mathbf{x}}_s^1). \tag{17}$$

Architecture. We use a DiT-based vector field Peebles & Xie (2023) conditioned on flow time, historical context $\mathcal{H}_p$, and future temporal features $\mathbf{y}^c$. The context encoder combines convolutions and Transformer-XL Dai et al. (2019) with relative positional embeddings, while flow-time embeddings control adaptive LayerNorm. Figures 1 and 5 illustrate the architecture.

3.3 Theoretical Motivation for Codebook-Prior Initialization

The benefit of codebook-prior initialization can be understood from the geometry of rectified transport in latent space. Let $\mathcal{P}_1$ denote the target latent distribution of $\mathbf{x}_s^1$, and let $\mathcal{P}_0$ denote an arbitrary source prior over the same latent space $\mathbb{R}^{L \times n_z}$. For any coupling $\Gamma \in \Pi(\mathcal{P}_0, \mathcal{P}_1)$, consider the rectified interpolation

$$\mathbf{x}_s^t = (1-t)\mathbf{x}_s^0 + t\mathbf{x}_s^1, \qquad t \in [0,1], \tag{18}$$

where $(\mathbf{x}_s^0, \mathbf{x}_s^1) \sim \Gamma$. We define the expected squared rectified transport cost under Γ as

$$\mathcal{C}_{\mathrm{rect}}(\Gamma) := \mathbb{E}_{(\mathbf{x}_s^0, \mathbf{x}_s^1) \sim \Gamma} \big[\|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F^2 \big]. \tag{19}$$

Proposition 3.1. *Assume $\mathcal{P}_0$ and $\mathcal{P}_1$ have finite second moments. Then the minimum expected squared rectified transport cost from $\mathcal{P}_0$ to $\mathcal{P}_1$ satisfies*

$$\inf_{\Gamma \in \Pi(\mathcal{P}_0, \mathcal{P}_1)} \mathcal{C}_{\mathrm{rect}}(\Gamma) = W_2^2(\mathcal{P}_0, \mathcal{P}_1), \tag{20}$$

where $W_2(\mathcal{P}_0, \mathcal{P}_1)$ is the 2-Wasserstein distance under the Frobenius ground cost.

Proposition 3.1 characterizes the minimum expected squared transport cost over all source–target couplings, providing a geometric motivation for source-prior design. ProtoFlow uses independent source–target sampling; the benefits of its prototype prior are evaluated empirically of Figure 4 . Let $\mathcal{P}_C := P_{\mathcal{Z}}$ denote ProtoFlow's sequence-level codebook prior, and let $\mathcal{P}_G = \mathcal{N}(0, I_{Ln_z})$ denote Gaussian initialization in the same latent space. If

$$W_2(\mathcal{P}_C, \mathcal{P}_1) < W_2(\mathcal{P}_G, \mathcal{P}_1), \tag{21}$$

then the codebook prior induces a smaller optimal rectified transport cost than Gaussian initialization. Under this assumption, this conditional result provides a geometric motivation for incorporating learned latent structure into the source prior. Detailed derivation is given in Appendix A.
This geometric motivation is complemented by Figure 4, where ProtoFlow achieves lower empirical Wasserstein distances between generated forecasts and observed targets, together with strong few-step forecasting performance. These results provide empirical support for the practical benefits of prototype-informed initialization.

4 EXPERIMENTS

Benchmarks. Our evaluation uses five widely adopted real-world benchmarks for high-dimensional MTS forecasting: Solar and Electricity Lai et al. (2018), Traffic and Taxi Salinas et al. (2019), and Wikipedia Gasthaus et al. (2019). These datasets span energy, transportation, and web-traffic applications, with sampling intervals ranging from 30 minutes to one day. Additional dataset details are provided in Appendix C.1.

Baselines. We compare ProtoFlow with strong generative forecasting baselines from the diffusion and flow families. For diffusion-based forecasting, we include TimeGrad Rasul et al. (2021), CSDI Tashiro et al. (2021), TSDiff Kollovieh et al. (2023), MG-TSD Fan et al. (2024), DyDiff Guo et al. (2025), and NsDiff Ye et al. (2025). We additionally consider the recent popular flow-based approachs FlowTime El-Gazzar & van Gerven (2025) and TSFlow Kollovieh et al. (2025). Detailed baseline descriptions are deferred to Appendix B.

Evaluation Metrics. We evaluate both probabilistic and deterministic forecasting quality. For probabilistic forecasting, we report the continuous ranked probability score on the aggregated series, $\mathrm{CRPS}_{\mathrm{sum}}$ Matheson & Winkler (1976). For deterministic accuracy, we report $\mathrm{NRMSE}_{\mathrm{sum}}$, which provides a scale-normalized comparison across datasets with different value ranges. Following standard probabilistic forecasting protocols, all results are computed from 100 generated samples.More details on the evaluation metrics are given in Appendix C.2.

Implementation Details. We use a history length of 96 and prediction lengths of $\{48, 96\}$, following high-dimensional time series forecasting settings Rasul et al. (2021); Tashiro et al. (2021). RevIN Kim et al. (2021) is applied except Solar. All experiments run on a single NVIDIA A100 GPU with 40 GB memory, with results averaged over three runs. More details are shown in Appendix Tables 6 and 7.

4.1 MAIN RESULTS

We evaluate ProtoFlow on five widely used high-dimensional MTS benchmarks with prediction horizons of 48 and 96 steps. As shown in Table 1, ProtoFlow achieves strong overall performance in both probabilistic forecasting ($\mathrm{CRPS}_{\mathrm{sum}}$) and point forecasting ($\mathrm{NRMSE}_{\mathrm{sum}}$), obtaining the best or joint-best results in 18 out of 20 comparisons, including comparisons with the Gaussian-prior variant ProtoFlow-G. Specifically,

	Electricity		Solar		Traffic		Taxi		Wikipedia	
Horizon+Method	$\mathrm{CRPS}_{\mathrm{sum}}$	$\mathrm{NRMSE}_{\mathrm{sum}}$	$\mathrm{CRPS}_{\mathrm{sum}}$	$\mathrm{NRMSE}_{\mathrm{sum}}$	$\mathrm{CRPS}_{\mathrm{sum}}$	$\mathrm{NRMSE}_{\mathrm{sum}}$	$\mathrm{CRPS}_{\mathrm{sum}}$	$\mathrm{NRMSE}_{\mathrm{sum}}$	$\mathrm{CRPS}_{\mathrm{sum}}$	$\mathrm{NRMSE}_{\mathrm{sum}}$
48 Steps										
• TimeGrad (2021)	0.031	0.044	0.324	0.736	0.043	0.072	0.252	0.334	0.082	0.105
• CSDI (2021)	0.029	0.050	0.328	0.742	-	-	-	-	-	-
• TSDiff (2023)	0.036	0.057	0.324	0.731	0.057	0.070	0.243	0.330	0.079	**0.091**
• MG-TSD (2024)	0.024	0.038	0.328	0.737	0.042	0.067	0.217	0.327	0.071	0.096
• FlowTime (2025)	0.033	0.052	0.321	0.729	0.054	0.077	0.419	0.514	0.091	0.115
• DyDiff (2025)	0.027	0.043	0.323	0.724	0.043	0.064	0.196	0.313	0.070	0.097
• NsDiff (2025)	0.025	0.040	0.311	0.703	<u>0.040</u>	**0.061**	<u>0.182</u>	0.299	0.074	<u>0.094</u>
• TSFlow (2025)	<u>0.015</u>	<u>0.036</u>	<u>0.289</u>	<u>0.681</u>	**0.038**	<u>0.064</u>	0.184	<u>0.294</u>	<u>0.065</u>	0.107
• ProtoFlow-G	0.014	0.031	0.301	0.689	0.035	0.057	0.153	0.279	0.067	0.118
• **ProtoFlow (ours)**	**0.013**	**0.028**	**0.275**	**0.626**	0.041	0.066	**0.150**	**0.272**	**0.063**	**0.091**
96 Steps										
• TimeGrad (2021)	0.039	0.047	0.347	0.749	0.046	0.079	0.369	0.454	0.109	0.138
• CSDI (2021)	0.047	0.044	0.330	0.753	-	-	-	-	-	-
• TSDiff (2023)	0.039	0.048	0.336	0.748	0.063	0.076	0.338	0.419	0.113	0.135
• MG-TSD (2024)	0.043	0.044	0.349	0.762	0.047	0.072	0.274	0.364	<u>0.094</u>	0.126
• FlowTime (2025)	0.048	0.059	0.328	0.751	0.069	0.083	0.474	0.576	0.135	0.163
• DyDiff (2025)	0.036	0.052	0.334	0.745	0.056	0.079	0.247	0.324	0.101	0.109
• NsDiff (2025)	0.033	0.047	0.309	0.714	0.047	0.069	0.236	<u>0.312</u>	0.099	<u>0.107</u>
• TSFlow (2025)	<u>0.025</u>	<u>0.044</u>	<u>0.299</u>	<u>0.703</u>	<u>0.044</u>	<u>0.068</u>	<u>0.212</u>	0.346	0.103	0.136
• ProtoFlow-G	0.019	0.037	0.305	0.717	0.036	0.058	0.159	0.287	0.096	0.114
• **ProtoFlow (ours)**	**0.015**	**0.022**	**0.284**	**0.638**	**0.034**	**0.054**	**0.157**	**0.281**	**0.092**	**0.101**

Table 1: The detailed results of ProtoFlow on the 48 and 96 prediction horizons. ProtoFlow-G: the variant of ProtoFlow that uses a standard Gaussian prior. - marks out-of-memory failures, Best and second-best forecasting results are bolded and underlined, respectively.

1) Compared with flow-based baselines, ProtoFlow reduces $\mathrm{CRPS}_{\mathrm{sum}}$ and $\mathrm{NRMSE}_{\mathrm{sum}}$ by an average of 13.6% and 17.4% over TSFlow, and by 42.6% and 34.4% over FlowTime, respectively. Compared with NsDiff, a strong diffusion-based baseline, ProtoFlow achieves average reductions of 22.0%

and 14.6%, respectively. All percentages are calculated by averaging the relative error reductions across the five datasets and two prediction horizons. These results demonstrate the effectiveness of ProtoFlow across diverse high-dimensional forecasting settings. **2)** ProtoFlow achieves strong sampling efficiency compared with competitive flow-based and diffusion-based methods. Specifically, it uses only 3 ODE sampling steps on all datasets except Wikipedia, which requires 5 steps. In comparison, TSFlow, a strong flow-based baseline, and NsDiff, a strong diffusion-based baseline, use 16 and 20 sampling steps, respectively, following the settings in their original papers. ProtoFlow achieves better overall forecasting accuracy than both methods with substantially fewer sampling steps, demonstrating a favorable balance between prediction quality and sampling efficiency. **3)** To examine the contribution of the prototype prior, we compare ProtoFlow with ProtoFlow-G, which uses a standard Gaussian prior during both training and inference. However, ProtoFlow converges in fewer training iterations (Figure 2 (a)), supporting the benefit of the prototype prior in accelerating flow matching training convergence.

4.2 Effects of latent representation and generation strategy.

We analyze the effects of latent representation and generation strategy in Table 2.

1) VQ-based variants outperform their VAE-based counterparts on both metrics across all five datasets under both AR and FM generation. Specifically, replacing VAE with VQ reduces the dataset-averaged CRPS_{sum} and $\text{NRMSE}_{\text{sum}}$ by 75.0% and 73.3% under AR generation, and by 57.3% and 51.9% under Gaussian-prior FM, respectively. These consistent improvements support the effectiveness of vector-quantized representations across different generation strategies. **2)** With VQ representations, ProtoFlow-G consistently outperforms VQ + AR across all five datasets, reducing the dataset-averaged CRPS_{sum} and $\text{NRMSE}_{\text{sum}}$ by 19.4% and 14.2%, respectively. By modeling the future latent sequence jointly, non-autoregressive FM avoids the discrepancy between conditioning on ground-truth preceding tokens during teacher-forced training and on generated tokens during AR inference. These results support the advantage of non-autoregressive FM in this setting. **3)** Overall, these results support the combination of vector-quantized representations and non-autoregressive flow matching. ProtoFlow further improves upon ProtoFlow-G on both metrics across all five datasets. Together with the earlier stabilization of its normalized training loss in Figure 2(a), these results support the benefit of incorporating learned prototype information into the FM source distribution.

Method	Electricity		Solar		Traffic		Taxi		Wikipedia		Avg.	
	C-S	NM-S	C-S	NM-S	C-S	NM-S	C-S	NM-S	C-S	NM-S	C-S	NM-S
VAE + AR	0.421	0.544	1.154	2.618	1.005	1.511	0.337	0.448	0.130	0.169	0.609	1.058
VAE + FM	0.129	0.213	0.643	1.324	0.323	0.564	0.227	0.293	0.117	0.128	0.288	0.504
VQ + AR	0.037	0.054	0.347	0.833	0.066	0.084	0.205	0.322	0.108	0.121	0.153	0.283
ProtoFlow-G	0.019	0.037	0.305	0.717	0.036	0.058	0.159	0.287	0.096	0.114	<u>0.123</u>	<u>0.243</u>
ProtoFlow	**0.015**	**0.022**	**0.284**	**0.638**	**0.034**	**0.054**	**0.157**	**0.281**	**0.092**	**0.101**	**0.116**	**0.219**

Table 2: The comparison results of 96 prediction horizons with baselines regarding CRPS_{sum} (C-S) and $\text{NRMSE}_{\text{sum}}$ (NM-S). AR: Auto-regressive Transformer and FM: Flow Matching .

4.3 Effects of the prototype prior on FM training/inference efficiency.

We smooth training losses with a five-epoch moving average and normalize each curve as $\widetilde{L}_e = (\bar{L}_e - L_{\text{end}})/(\bar{L}_1 - L_{\text{end}})$, where $\bar{L}_e$ denotes the smoothed loss at epoch e, and L_{end} is the median of the smoothed losses over the final 5% of training epochs for the corresponding run. Figure 2(a) shows that ProtoFlow approaches its own terminal loss level in fewer epochs than ProtoFlow-G and TSFlow across all three datasets. The comparison with ProtoFlow-G, which shares the same representation and generation architecture, supports earlier FM training stabilization with prototype initialization. These curves describe relative loss reduction toward each run's observed endpoint and do not establish equivalent absolute optimization errors or predictive accuracy across methods.

Figure 2 (b) reports inference speedups of 3.66–$3.90\times$ over TSFlow. ProtoFlow combines 3-step ODE sampling with non-autoregressive latent generation, avoiding autoregressive dependencies, whereas TSFlow uses 16 steps following its original configuration. These latency gains accompany a larger training-memory footprint (4.08–$7.49\times$),associated with the additional components used for latent representation learning and generation. Meanwhile, ProtoFlow outperforms TSFlow on both

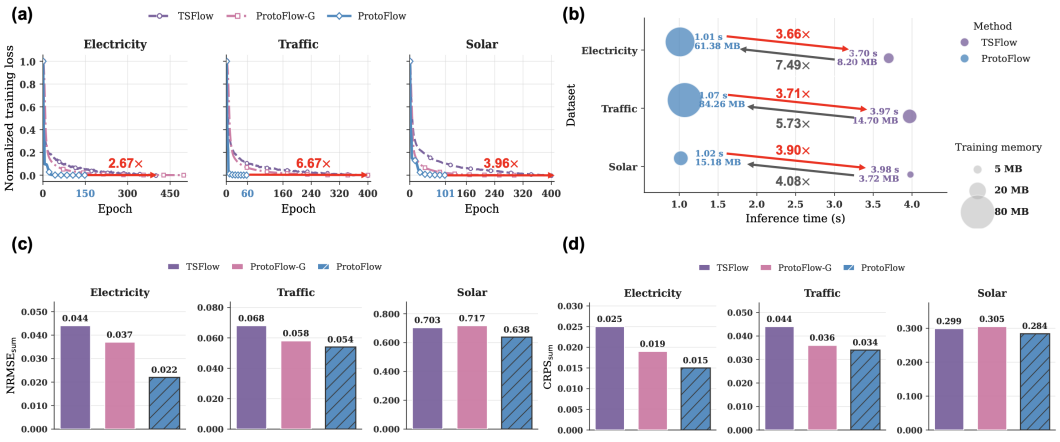

Figure 2: Comparison of training convergence (**a**), inference efficiency, training memory (**b**), and forecasting performance (**c-d**) on Electricity, Traffic, and Solar with a prediction length of 96.

forecasting metrics across all three datasets and generally improves upon ProtoFlow-G (Figures 2 (c)–(d)).

Method	Electricity		Solar		Traffic		Taxi		Wikipedia		Avg.	
	C-S	NM-S	C-S	NM-S	C-S	NM-S	C-S	NM-S	C-S	NM-S	C-S	NM-S
ProtoFlow-G	0.019	0.037	0.305	0.717	0.036	0.058	0.159	0.287	0.096	0.114	0.123	0.243
ProtoFlow-uniform	0.018	0.030	0.356	0.769	0.036	0.056	**0.157**	0.284	0.099	0.108	0.133	0.249
ProtoFlow	**0.015**	**0.022**	**0.284**	**0.638**	**0.034**	**0.054**	**0.157**	**0.281**	**0.092**	**0.101**	**0.116**	**0.219**

Table 3: The forecasting performance with frequency-weighted prototype (ProtoFlow), uniform-weight prototype (ProtoFlow-uniform), and isotropic Gaussian priors (ProtoFlow-G) at a prediction length of 96.

4.4 Effects of codebook size and prototype frequency information.

The prototype prior provides a data-informed initialization for flow matching through learned latent structure. We examine codebook size, which shapes latent quantization and the mixture prior, and prototype frequency information, which determines probability allocation across prototypes. Specifically, we vary $K \in \{64, 128, 256, 512\}$ with local covariance rank $r = 2$ fixed across datasets, and compare frequency-based mixing weights with uniform weights. Figure 3, Table 3 and 4 report their respective effects on forecasting performance.

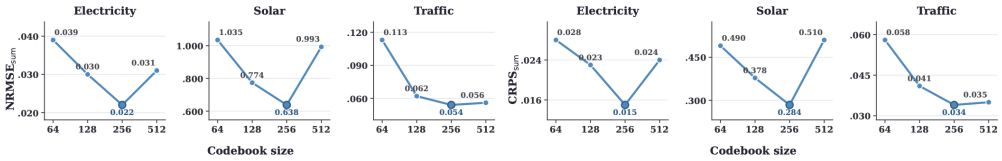

Figure 3: Effects of codebook size on ProtoFlow at the 96-step prediction length.

Figure 3 shows that both metrics improve from $K = 64$ to 256, which yields the best observed results on all three datasets. At $K = 512$, errors rise substantially on Electricity and Solar but only slightly on Traffic. Larger codebooks may leave fewer samples per prototype for estimating local prior statistics, suggesting a balance between representational capacity and reliable prior estimation. Then, we replace the mixing weights estimated from training statistics with $\pi_k = 1/K$ in Eq. equation 9, yielding $p_c^{\text{uniform}}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} p_k(\mathbf{x}; \mu_k, B_k, \sigma_k)$. Table 3 compares ProtoFlow-G, ProtoFlow-uniform, and ProtoFlow using respective ODE step counts of $\{10, 50, 10, 20, 10\}$, $\{5, 50, 5, 10, 10\}$, and $\{3, 3, 3, 3, 5\}$, following the dataset order in the table. ProtoFlow achieves the best or joint-best performance on both metrics across all five datasets while using fewer steps. Although uniform weighting remains competitive in several settings, its larger sampling budgets offer no overall accuracy advantage. These findings support combining learned prototype structure with empirical frequency information for accurate few-step forecasting.

	Electricity			Solar			Traffic		
	Frequency	Uniform	Stage1 MSE	Frequency	Uniform	Stage1 MSE	Frequency	Uniform	Stage1 MSE
Coodebook size									
64	0.039	0.044	7.76e-4	1.035	0.962	7.70e-4	0.113	0.105	5.82e-4
128	0.030	0.032	7.13e-4	0.774	0.761	6.75e-4	0.062	0.066	5.23e-4
256	0.022	0.030	6.02e-4	0.638	0.769	6.13e-4	0.054	0.056	4.78e-4
512	0.031	0.033	6.01e-4	0.993	0.758	6.18e-4	0.056	0.057	4.69e-4

Table 4: Effects of codebook size on Stage 1 reconstruction MSE and $\text{NRMSE}_{\text{sum}}$ under frequency and uniform weighting at a 96-step prediction length.

Table 4 shows that $K = 64$ yields the highest reconstruction errors (red cells) and forecasting errors under both weighting schemes across all three datasets, suggesting that insufficient representational capacity can limit forecasting quality. Increasing K from 256 to 512 provides little additional reconstruction benefit (yellow cells), yet forecasting errors increase under frequency weighting, particularly on Solar, and decrease slightly under uniform weighting. One possible explanation is that additional prototypes subdivide existing latent patterns without substantial representational gains, while the resulting probability allocation may make conditional transport more difficult to learn. The contrasting trends under the two weighting schemes suggest an interaction between codebook capacity and source probability allocation. Given the available training data and a fixed ODE sampling budget, these design choices should therefore be considered jointly, as improved reconstruction alone does not guarantee better forecasting.

4.5 Effects of the prototype prior on transport and sampling-step sensitivity

Figure 4 examines both the transport geometry and sampling-step sensitivity of ProtoFlow and ProtoFlow-G across six ODE steps on three benchmarks with a prediction horizon of 96. ProtoFlow consistently exhibits a smaller W_2 distance than ProtoFlow-G across all sampling budgets, indicating that the prototype-based initialization induces a transport configuration that is better aligned with the target distribution. This observation is consistent with the theoretical motivation in Section 3.3, where a source prior closer to the target distribution corresponds to a smaller optimal rectified transport cost. ProtoFlow also achieves its best or near-best CRPS_{sum} with only a few ODE steps and remains relatively stable as the sampling budget increases.

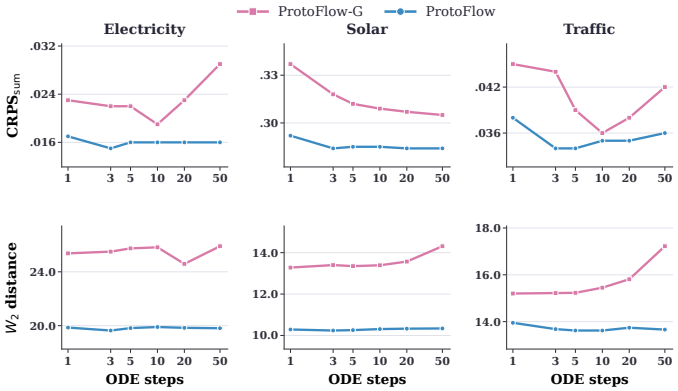

Figure 4: Effect of ODE sampling steps on 96-step probabilistic forecasting performance and W_2 transport distance for ProtoFlow and ProtoFlow-G.

5 Conclusion

In this paper, we propose ProtoFlow, a prototype-prior flow matching framework that integrates vector-quantized representations with non-autoregressive transport for high-dimensional multivariate time series forecasting. By constructing the source distribution from learned prototypes and their local statistics, ProtoFlow achieves strong forecasting performance with few ODE steps, alongside earlier stabilization of flow matching training and reduced sensitivity to the sampling budget. Beyond these empirical gains, our findings highlight a broader role for learned codebooks: their structure can inform both the representation of time series and the initialization of generative transport. This perspective motivates closer integration of representation learning and source-prior design as a promising direction for efficient generative time series forecasting.

AI USE STATEMENT

Generative AI tools assisted with identifying relevant literature, drafting and refining manuscript text, LaTeX formatting, providing feedback on experimental design, and supporting code development. The authors reviewed all AI-assisted content, checked scientific claims against primary sources and experimental evidence, and retain full responsibility for the accuracy and integrity of the final manuscript.

REPRODUCIBILITY STATEMENT

Section 3 describes the vector-quantized tokenizer, prototype-prior construction, training objectives, and inference procedure. The theoretical motivation is presented in Section 3.3, with further derivations in Appendix A. Section 4 specifies the benchmarks, baselines, evaluation settings, and ablation studies. Additional dataset information and metric definitions are provided in Appendices C.1 and C.2, respectively. Stage-specific model configurations and training hyperparameters are reported in Tables 6 and 7. Experiments use a single NVIDIA A100 GPU with 40 GB memory, and results are averaged over three runs, with forecasting metrics computed from 100 generated samples per prediction window. The code will be made publicly available upon acceptance.

REFERENCES

Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, et al. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21(116):1–6, 2020.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Defu Cao, Wen Ye, Yizhou Zhang, and Yan Liu. Timedit: General-purpose diffusion transformers for time series foundation model. *arXiv preprint arXiv:2409.02322*, 2024.

Zhicheng Chen, FENG SHIBO, Zhong Zhang, Xi Xiao, Xingyu Gao, and Peilin Zhao. Sdformer: Similarity-driven discrete transformer for time series generation. *Advances in Neural Information Processing Systems*, 37:132179–132207, 2024.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2978–2988, 2019.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Ahmed El-Gazzar and Marcel van Gerven. Probabilistic forecasting via autoregressive flow matching. *arXiv preprint arXiv:2503.10375*, 2025.

Xinyao Fan, Yueying Wu, Chang Xu, Yuhao Huang, Weiqing Liu, and Jiang Bian. Mg-tsd: Multi-granularity time series diffusion models with guided learning process. *arXiv preprint arXiv:2403.05751*, 2024.

Shibo Feng, Peilin Zhao, Liu Liu, Pengcheng Wu, and Zhiqi Shen. Hdt: Hierarchical discrete transformer for multivariate time series forecasting. *arXiv preprint arXiv:2502.08302*, 2025.

Shibo Feng, Zhicheng Chen, Xi Xiao, Zhong Zhang, Qing Li, Xingyu Gao, and Peilin Zhao. Msdformer: Multi-scale discrete transformer for time series generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

Jan Gasthaus, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. Probabilistic forecasting with spline quantile function rnns. In *The 22nd international conference on artificial intelligence and statistics*, pp. 1901–1910. PMLR, 2019.

Xingzhuo Guo, Yu Zhang, Baixu Chen, Haoran Xu, Jianmin Wang, and Mingsheng Long. Dynamical diffusion: Learning temporal dynamics with diffusion models. *arXiv preprint arXiv:2503.00951*, 2025.

Hongbin Huang, Minghua Chen, and Xiao Qiao. Generative learning for financial time series with irregular and scale-invariant patterns. In *The Twelfth International Conference on Learning Representations*, 2024.

Jiawei Jiang, Chengkai Han, Wayne Xin Zhao, and Jingyuan Wang. Pdformer: Propagation delay-aware dynamic long-range transformer for traffic flow prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 4365–4373, 2023.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International conference on learning representations*, 2021.

Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

Marcel Kollovieh, Abdul Fatir Ansari, Michael Bohlke-Schneider, Jasper Zschiegner, Hao Wang, and Yuyang Bernie Wang. Predict, refine, synthesize: Self-guiding diffusion models for probabilistic time series forecasting. *Advances in Neural Information Processing Systems*, 36:28341–28364, 2023.

Marcel Kollovieh, Marten Lienen, David Lüdke, Leo Schwinn, and Stephan Günnemann. Flow matching with gaussian process priors for probabilistic time series forecasting. *arXiv preprint arXiv:2410.03024*, 2024.

Marcel Kollovieh, Marten Lienen, David Lüdke, Leo Schwinn, and Stephan Günnemann. Flow matching with gaussian process priors for probabilistic time series forecasting. In *International Conference on Learning Representations*, volume 2025, pp. 95435–95457, 2025.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 95–104, 2018.

Daesoo Lee, Sara Malacarne, and Erlend Aune. Vector quantized time series generation with a bidirectional prior model. *arXiv preprint arXiv:2303.04743*, 2023.

Wei Li, Shibo Feng, Pengcheng Wu, Xingyu Gao, Min Wu, and Peilin Zhao. Sdflow: Similarity-driven flow matching for time series generation. *arXiv preprint arXiv:2605.05736*, 2026.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

James E Matheson and Robert L Winkler. Scoring rules for continuous probability distributions. *Management science*, 22(10):1087–1096, 1976.

Caspar Meijer and Lydia Y Chen. The rise of diffusion models in time-series forecasting. *arXiv preprint arXiv:2401.03006*, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International conference on machine learning*, pp. 8857–8868. PMLR, 2021.

Kashif Rasul, Andrew Bennett, Pablo Vicente, Umang Gupta, Hena Ghonia, Anderson Schneider, and Yuriy Nevmyvaka. Vq-tr: Vector quantized attention for time series forecasting. In *International Conference on Learning Representations*, volume 2024, pp. 47137–47158, 2024.

Weilin Ruan, Siru Zhong, Haomin Wen, and Yuxuan Liang. Vision-enhanced time series forecasting via latent diffusion models. *arXiv preprint arXiv:2502.14887*, 2025.

David Salinas, Michael Bohlke-Schneider, Laurent Callot, Roberto Medico, and Jan Gasthaus. High-dimensional multivariate forecasting with low-rank gaussian copula processes. *Advances in neural information processing systems*, 32, 2019.

Florian Schmidt. Generalization in generation: A closer look at exposure bias. *arXiv preprint arXiv:1910.00292*, 2019.

Zezhi Shao, Zhao Zhang, Wei Wei, Fei Wang, Yongjun Xu, Xin Cao, and Christian S Jensen. Decoupled dynamic spatial-temporal graph neural network for traffic forecasting. *arXiv preprint arXiv:2206.09112*, 2022.

Lifeng Shen and James Kwok. Non-autoregressive conditional diffusion models for time series prediction. In *International Conference on Machine Learning*, pp. 31016–31029. PMLR, 2023.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. pmlr, 2015.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Sabera Talukder, Yisong Yue, and Georgia Gkioxari. Totem: Tokenized time series embeddings for general time series analysis. *arXiv preprint arXiv:2402.16412*, 2024.

Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in neural information processing systems*, 34:24804–24816, 2021.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *arXiv preprint arXiv:2302.00482*, 2023.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Daoyu Wang, Mingyue Cheng, Zhiding Liu, and Qi Liu. Timedart: A diffusion autoregressive transformer for self-supervised time series representation. *arXiv preprint arXiv:2410.05711*, 2024.

Haixu Wu, Hang Zhou, Mingsheng Long, and Jianmin Wang. Interpretable weather forecasting for worldwide stations with a unified deep model. *Nature Machine Intelligence*, 5(6):602–611, 2023.

Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, Ying Wei, and Junzhou Huang. Adversarial sparse transformer for time series forecasting. *Advances in neural information processing systems*, 33: 17105–17115, 2020.

Weiwei Ye, Zhuopeng Xu, and Ning Gui. Non-stationary diffusion for probabilistic time series forecasting. *arXiv preprint arXiv:2505.04278*, 2025.

Xi Nicole Zhang, Yuan Pu, Yuki Kawamura, Andrew Loza, Yoshua Bengio, Dennis Shung, and Alexander Tong. Trajectory flow matching with applications to clinical time series modelling. *Advances in Neural Information Processing Systems*, 37:107198–107224, 2024.

Supplementary Materials for ProtoFlow

A Additional Theoretical Analysis of Codebook-Prior Initialization

In this section, we provide the formal derivation underlying the theoretical motivation for codebook-prior initialization in ProtoFlow.

A.1 Optimal Rectified Transport Cost and Wasserstein Distance

Let $\mathcal{P}_1$ denote the target latent distribution of $\mathbf{x}_s^1$, and let $\mathcal{P}_0$ be an arbitrary source prior defined on the same latent space $\mathbb{R}^{L\times n_z}$. Assume that both $\mathcal{P}_0$ and $\mathcal{P}_1$ have finite second moments. For any coupling

$$\Gamma \in \Pi(\mathcal{P}_0, \mathcal{P}_1),$$

where $\Pi(\mathcal{P}_0, \mathcal{P}_1)$ denotes the set of all joint distributions with marginals $\mathcal{P}_0$ and $\mathcal{P}_1$, define the rectified interpolation path

$$\mathbf{x}_s^t = (1-t)\mathbf{x}_s^0 + t\mathbf{x}_s^1, \qquad t \in [0,1]. \tag{22}$$

The expected squared rectified transport cost under coupling Γ is defined as

$$\mathcal{C}_{\text{rect}}(\Gamma) := \mathbb{E}_{(\mathbf{x}_s^0, \mathbf{x}_s^1)\sim\Gamma}\left[\|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F^2\right]. \tag{23}$$

We further define the minimum expected squared rectified transport cost from $\mathcal{P}_0$ to $\mathcal{P}_1$ as

$$\mathcal{C}_{\text{opt}}(\mathcal{P}_0, \mathcal{P}_1) := \inf_{\Gamma\in\Pi(\mathcal{P}_0,\mathcal{P}_1)} \mathcal{C}_{\text{rect}}(\Gamma). \tag{24}$$

Proposition A.1. *Under the above definitions,*

$$\mathcal{C}_{\text{opt}}(\mathcal{P}_0, \mathcal{P}_1) = W_2^2(\mathcal{P}_0, \mathcal{P}_1), \tag{25}$$

where $W_2(\mathcal{P}_0, \mathcal{P}_1)$ is the 2-Wasserstein distance under the Frobenius ground cost.

Proof. For a fixed pair $(\mathbf{x}_s^0, \mathbf{x}_s^1)$, the rectified path is given by Eq. equation 22. Differentiating with respect to t gives

$$\frac{d\mathbf{x}_s^t}{dt} = \mathbf{x}_s^1 - \mathbf{x}_s^0. \tag{26}$$

Since the velocity is constant along the rectified path, its path length is

$$\int_0^1 \left\| \frac{d\mathbf{x}_s^t}{dt} \right\|_F dt = \int_0^1 \|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F dt = \|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F. \tag{27}$$

Hence, the squared rectified path length is

$$\|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F^2. \tag{28}$$

Taking expectation under a coupling $\Gamma \in \Pi(\mathcal{P}_0, \mathcal{P}_1)$ gives

$$\mathcal{C}_{\text{rect}}(\Gamma) = \mathbb{E}_{(\mathbf{x}_s^0, \mathbf{x}_s^1)\sim\Gamma}\left[\|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F^2\right]. \tag{29}$$

By definition, the squared 2-Wasserstein distance between $\mathcal{P}_0$ and $\mathcal{P}_1$ under the Frobenius ground cost is

$$W_2^2(\mathcal{P}_0, \mathcal{P}_1) = \inf_{\Gamma\in\Pi(\mathcal{P}_0,\mathcal{P}_1)} \mathbb{E}_{(\mathbf{x}_s^0, \mathbf{x}_s^1)\sim\Gamma}\left[\|\mathbf{x}_s^1 - \mathbf{x}_s^0\|_F^2\right]. \tag{30}$$

Combining this with Eq. equation 24 yields

$$\mathcal{C}_{\text{opt}}(\mathcal{P}_0, \mathcal{P}_1) = W_2^2(\mathcal{P}_0, \mathcal{P}_1).$$

□

A.2 Implication for Codebook-Prior Initialization

Let $\mathcal{P}_C$ denote the source distribution induced by the learned codebook prior used in ProtoFlow, and let

$$\mathcal{P}_G := \mathcal{N}(0, I_{Ln_z}) \tag{31}$$

denote the Gaussian prior in the same latent space.

Corollary A.2. *If the learned codebook prior is closer to the target latent distribution than the Gaussian prior in* 2*-Wasserstein distance, namely*

$$W_2(\mathcal{P}_C, \mathcal{P}_1) < W_2(\mathcal{P}_G, \mathcal{P}_1), \tag{32}$$

then

$$\mathcal{C}_{\text{opt}}(\mathcal{P}_C, \mathcal{P}_1) < \mathcal{C}_{\text{opt}}(\mathcal{P}_G, \mathcal{P}_1). \tag{33}$$

Thus, under optimal source–target coupling, the codebook prior yields a smaller expected squared path length for the linear interpolation.

Proof. By Proposition A.1,

$$\mathcal{C}_{\text{opt}}(\mathcal{P}_C, \mathcal{P}_1) = W_2^2(\mathcal{P}_C, \mathcal{P}_1), \tag{34}$$

and

$$\mathcal{C}_{\text{opt}}(\mathcal{P}_G, \mathcal{P}_1) = W_2^2(\mathcal{P}_G, \mathcal{P}_1). \tag{35}$$

Under the assumption in Eq. equation 32, and since both Wasserstein distances are nonnegative, squaring preserves the strict inequality:

$$W_2^2(\mathcal{P}_C, \mathcal{P}_1) < W_2^2(\mathcal{P}_G, \mathcal{P}_1). \tag{36}$$

Substituting the above equalities gives Eq. equation 33. □

Corollary A.2 shows that the advantage of codebook-prior initialization can be interpreted geometrically: whenever the learned codebook prior is closer to the target latent distribution than Gaussian initialization in Wasserstein distance, it yields a smaller optimal rectified transport cost.

B Related Work

We discuss two lines of research underlying ProtoFlow: vector-quantized latent modeling and diffusion/flow matching for multivariate time series forecasting.

Vector-quantized Latent Modeling. Vector quantization Van Den Oord et al. (2017) maps continuous representations to a finite set of learned embeddings, enabling discrete latent modeling. Its promising performance in time series forecasting and generation has motivated growing interest in this representation strategy. For example, in time series forecasting task, VQ-TR Rasul et al. (2024) introduces quantization within attention for computationally efficient forecasting and HDT Feng et al. (2025) formulates multivariate forecasting through hierarchical discrete token modeling. Related generation methods explore diverse latent structures and generation strategies within VQ-based time series modeling. TimeVQVAE Lee et al. (2023) employs bidirectional prior modeling, while SDformer Chen et al. (2024) combines similarity-driven vector quantization with autoregressive or masked token modeling. MSDformer Feng et al. (2026) extends discrete modeling through multi-scale tokenization and autoregressive generation. SDFlow Li et al. (2026) instead performs non-autoregressive generation through variational flow matching in a frozen VQ latent space, incorporating categorical supervision over codebook indices. In addition, TOTEM Talukder et al. (2024) learns a shared vector-quantized tokenizer across diverse domains, enabling transferable discrete representations for forecasting, imputation, and anomaly detection tasks. ProtoFlow additionally uses the codebook as a source of distributional information, constructing a prototype prior for continuous transport in the latent embedding space.

Diffusion and Flow Matching for Forecasting. Diffusion-based methods have demonstrated strong performance in time series analysis, particularly in multivariate forecasting, where iterative denoising enables flexible modeling of future uncertainty. TimeGrad Rasul et al. (2021) combines diffusion with autoregressive prediction, while CSDI Tashiro et al. (2021) and TimeDiff Shen & Kwok (2023) investigate conditional reconstruction and non-autoregressive prediction, respectively.

TSDiff Kollovieh et al. (2023) enables forecasting with an unconditional diffusion model through inference-time guidance. Subsequent methods incorporate multi-granularity supervision MG-TSD Fan et al. (2024), temporal dynamics DyDiff Guo et al. (2025), and time-varying uncertainty NsDiff Ye et al. (2025). Although effective, iterative sampling can impose substantial inference costs.

Flow matching learns transport from a source distribution to the data distribution through vector-field regression, enabling ODE-based generation that can support accurate sampling with fewer integration steps through suitable probability-path design Lipman et al. (2022); Tong et al. (2023). For time series forecasting task, FlowTime El-Gazzar & van Gerven (2025) develops autoregressive flow matching in observation space, whereas TSFlow Kollovieh et al. (2025) incorporates Gaussian-process priors to encode temporal structure and supports both conditional and unconditional modeling. These approaches motivate studying both transport dynamics and source distributions. ProtoFlow explores this direction by constructing a source prior from learned latent prototypes, supporting accurate forecasting with few ODE sampling steps while facilitating earlier stabilization of flow matching training.

Positioning of ProtoFlow. ProtoFlow connects vector-quantized representations with prototype-prior flow matching. Its codebook serves two roles: representing target sequences and defining a structured source distribution through prototype centers, empirical frequencies, and local variability. Conditioned on historical context, a non-autoregressive vector field jointly transports the future latent sequence, which is then decoded into forecasts. **The distinguishing design is the reuse of learned representation structure to initialize latent transport, providing a data-informed source prior that supports earlier FM training stabilization and accurate forecasting with few sampling steps.** Our experiments examine how this initialization affects FM training dynamics, few-step forecasting, and sensitivity to the sampling budget.

C More Implementation Details

C.1 More Details on Datasets

We conduct extensive evaluations of the proposed ProtoFlow framework on five widely-used real-world benchmark datasets for multivariate time series forecasting, covering representative application domains Lai et al. (2018) including *Energy* (Solar and Electricity), *Traffic* (Traffic and Taxi), and *Society and Economics* (Wikipedia). Table 5 summarizes the key characteristics of these datasets. All experiments are repeated three times, implemented in PyTorch Paszke et al. (2019) and GluonTS Alexandrov et al. (2020).

DATASET	Dimension	Domain	Freq	Total Time Steps	Pred Length	Rolling Windows
SOLAR	137	$\mathbb{R}^+$	Hourly	7,009	$\{48, 96\}$	7
ELECTRICITY	370	$\mathbb{R}^+$	Hourly	5,790	$\{48, 96\}$	7
TRAFFIC	963	(0,1)	Hourly	10,413	$\{48, 96\}$	7
TAXI	1214	$\mathbb{N}$	30-Min	1,488	$\{48, 96\}$	57
WIKIPEDIA	2000	$\mathbb{N}$	Daily	792	$\{48, 96\}$	5

Table 5: The dataset properties used in our experiments are summarized as follows.

Specifically, the sources of these datasets are shown in the following:

- **SOLAR** records 10-minute solar power production from 137 PV plants in Alabama in 2006;
- **ELECTRICITY** comprises electricity consumption for $n = 370$ clients measured at 15-minute intervals during 2012–2014;
- **TRAFFIC** provides hourly road occupancy rates (normalized to $[0, 1]$) collected by highway sensors in the San Francisco Bay Area over multiple years;
- **TAXI** is a spatio-temporal series of New York City taxi rides aggregated every 30 minutes at 1214 locations, using January 2015 for training and January 2016 for testing as shown in GluonTS Alexandrov et al. (2020);
- **WIKIPEDIA** contains daily page-view counts for 2000 Wikipedia pages of GluonTS repository Alexandrov et al. (2020);

Following standard practice, we adopt a strictly chronological split based on a fixed cutoff time, partitioning each dataset into 60% training, 20% validation, and 20% test. We further evaluate with a rolling-window protocol on the test segment: forecasts are initiated from multiple start times that are evenly spaced after the last training timestamp, and metrics are aggregated across windows. For hourly datasets, we use 7 rolling windows; for the remaining datasets we use 5 windows, except for Taxi where 57 windows are employed to cover the full test period.

C.2 More Evaluation Protocols

We evaluate multivariate forecasting using CRPS_{sum} and $\text{NRMSE}_{\text{sum}}$, which measure probabilistic and point forecasting accuracy on the series obtained by summing across variables at each time step. These metrics assess how well predictions capture the aggregate behavior of the **multivariate time series setting**. Below, we describe their computation using $N = 100$ generated forecast samples and report dataset-specific hyperparameters for Stage 1 and Stage 2.

Metrics. CRPS measures the compatibility of a cumulative distribution function P with an observation x as: $\text{CRPS}(\mathcal{F}, x) = \int_{\mathbb{R}} (P(y) - \mathbb{I}\{x \leq y\})^2 dy$, where $\mathbb{I}\{x \leq y\}$is the indicator function which is one if $x \leq y$ and zero otherwise. The empirical CDF of P, i.e., $\hat{P}(y) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{X_i \leq y\}$ with n samples $X_i \sim P$ as the approximation of the predictive CDF. It utilizes N samples to estimate the empirical CDF and take the CRPS_{sum} in the multivariate case.

$$\text{CRPS}_{\text{sum}} = \mathbb{E}_t \left[\text{CRPS} \left(\widehat{P}_{\text{sum}}(t), \sum_i x_i^t \right) \right]. \tag{37}$$

The Normalized Root Mean Squared Error (NRMSE) is a standardized version of the Root Mean Squared Error (RMSE) that accounts for the scale of the target values. The formula for NRMSE is given below:

$$\text{NRMSE} = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \left(\frac{y_t - \hat{y}_t}{y_{max} - y_{min}} \right)^2}, \tag{38}$$

where $\hat{y}_t$ represents the predicted target, and y_t represents the true target. y_{max} and y_{min} are the minimum and maximum of the measured target values, respectively. The NRMSE quantifies the average squared discrepancy between the predictions and actual observations, normalized by the range of the target values and $\text{NRMSE}_{\text{sum}}$ is the sum of the NRMSE in our 100 samples experimental setting. A lower NRMSE indicates higher predictive accuracy.

In this section, we report the detailed hyperparameter settings of ProtoFlow for both Stage 1 and Stage 2. In Table 6, we list the codebook size and codebook dimension of the vector-quantization strategy for each dataset, together with the EMA decay coefficient (λ) used to smooth codebook updates and the coefficient (β) in Eqn. 8, which controls the trade-off between the commitment loss and the reconstruction loss.

Datasets	Codebook Size	Codebook dim	Enc/Dec Layers	λ	β
Solar	256	128	2	0.99	0.1
Electricity	256	256	2	0.99	0.1
Traffic	256	256	2	0.99	0.1
Taxi	256	512	2	0.99	0.05
Wikipedia	256	512	2	0.99	0.05

Table 6: Detailed hyperparameters of Stage 1.

For Stage 2, we report the number of layers and hidden dimension of DiT_θ, the training batch size and learning rate, and the number of ODE sampling steps used at inference, which are shown in Table 7.

C.3 The Details of ProtoFlow

This section provides comprehensive details of ProtoFlow, including the architectural design of Stage 1 and Stage 2 and the full training and inference procedures.

Datasets	Hidden dim	DiT$_\theta$ layers	Batch size	Steps	lr
Solar	128	4	64	3	5e-4
Electricity	256	4	64	3	5e-4
Traffic	256	5	64	3	3e-4
Taxi	512	5	64	3	3e-4
Wikipedia	512	4	64	5	3e-4

Table 7: Detailed hyperparameters of Stage 2.

Overview. We first describe the model architecture of ProtoFlow, including the design of the vector-quantized latent space (Stage 1) and the flow matching generator (Stage 2). We then provide pseudocode for the training and inference procedures.

Stage 1: Vector-Quantized Latent Representation. We detail the architectures of the Stage 1 model in Tables 8 and 9. Stage 1 consists of an encoder, a vector quantization module, and a decoder, which together learn a compact discrete latent representation of the input time series. The tables summarize the network structure, codebook design, and training configurations.

Layer	Function	Descriptions
1	Convolution	input channel=d, output channel=D, kernel size=3, stride=1, padding=1
2	ReLU	nn.ReLU()
3	Convolution	input channel=D, output channel=D, kernel size=4, stride=2, padding=1
4	ResNet	input channel=D, depth=3, dilation growth rate=3
5	ReLU	nn.ReLU()
6	Convolution	input channel=D, output channel=D, kernel size=4, stride=2, padding=1
7	ResNet	input channel=D, depth=3, dilation growth rate=3
8	ReLU	nn.ReLU()
9	Convolution	input channel= d_c, output channel=H, kernel size=3, stride=1, padding=1

Table 8: The detailed architecture of the time serie tokenizer's encoder.

Layer	Function	Descriptions
1	Convolution	input channel=d_c, output channel=D, kernel size=3, stride=1, padding=1
2	ReLU	nn.ReLU()
3	ResNet	input channel=D, depth=3, dilation growth rate=3
4	ReLU	nn.ReLU()
5	Upsample	nn.Upsample()
6	Convolution	input channel=D, output channel=D, kernel size=3, stride=1, padding=1
7	ResNet	input channel=D, depth=3, dilation growth rate=3
8	ReLU	nn.ReLU()
9	Upsample	nn.Upsample()
10	Convolution	input channel=D, output channel=D, kernel size=3, stride=1, padding=1
11	ReLU	nn.ReLU()
12	Convolution	input channel=D, output channel=D, kernel size=3, stride=1, padding=1
13	ReLU	nn.ReLU()
14	Convolution	input channel=D, output channel=d, kernel size=3, stride=1, padding=1

Table 9: The detailed architecture of the time serie tokenizer's decoder.

Stage 2: Flow Matching with DiT$_\theta$. The architecture of the flow matching generator is illustrated in Figure 5. The figure provides a detailed view of the transformer blocks, conditioning mechanisms, and adaptive normalization used in Stage 2.

Training and Inference Procedures. We present the pseudocode of training and inference procedures for ProtoFlow in Algorithm 1, 2, and 3.

Codebook Reset Strategy in Stage 1. We maintain EMA statistics of assignment counts and assigned feature sums for each codebook entry. Let u_k denote the smoothed assignment count and

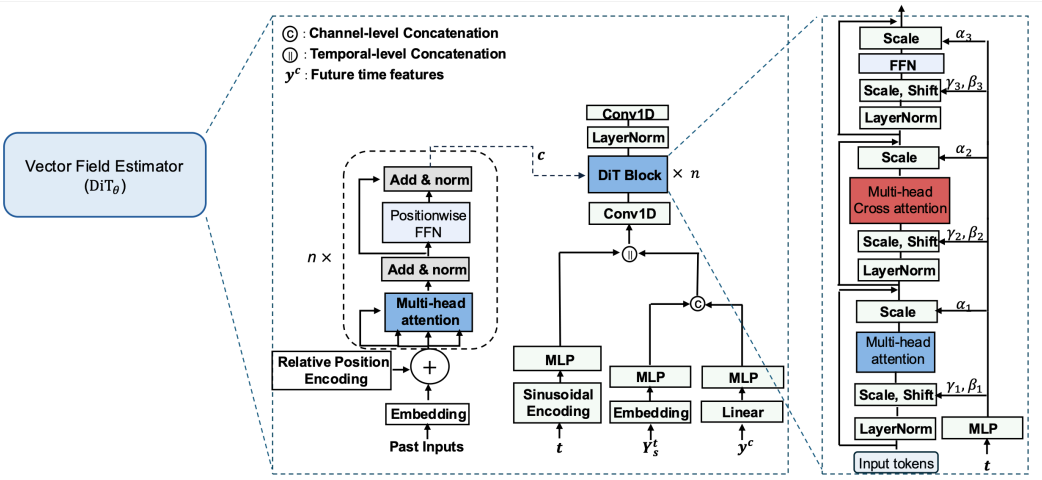

Figure 5: Illustration of model architecture of DiT$_\theta$ of Stage2.

$\mathcal{Z}_k^{\mathrm{EMA}}$ the normalized EMA feature sum. To mitigate codebook collapse, we reinitialize underutilized entries as

$$\mathcal{Z}_k \leftarrow \begin{cases} h_k^{\mathrm{reset}}, & u_k < u_{\mathrm{th}}, \\ \mathcal{Z}_k^{\mathrm{EMA}}, & u_k \geq u_{\mathrm{th}}, \end{cases} \tag{39}$$

where $u_{\mathrm{th}} = 1$ and h_k^{reset} is a normalized replacement candidate from the current batch. Candidates are constructed from the first K flattened encoder outputs. If fewer than K are available, we repeat them and add Gaussian noise with per-coordinate standard deviation $0.01/\sqrt{n_z}$ before selecting and normalizing the first K vectors. Resetting replaces the codebook entries while preserving their EMA statistics and is performed only during Stage 1 training.

Algorithm 1 ProtoFlow Stage 1: Vector-Quantized Tokenizer Training

Input: Training target sequences $\{\mathcal{X}_t\}$
Parameters: Codebook size K, embedding dimension n_z, downsampling ratio ρ, commitment weight β
Output: Encoder $\mathcal{E}_\psi$, decoder $\mathcal{G}_\phi$, codebook $\mathcal{Z}$
Initialize $\mathcal{E}_\psi$, $\mathcal{G}_\phi$, and unit-normalized codebook $\mathcal{Z}$
while training is not complete **do**
  Sample a minibatch $\{\mathcal{X}_{t,b}\}_{b\in\mathcal{B}}$
  Encode and normalize latent vectors: $\hat{z}_b \leftarrow \mathrm{Norm}(\mathcal{E}_\psi(\mathcal{X}_{t,b}))$
  For all b, ℓ, assign $a_{b,\ell} \leftarrow \arg\max_k \langle \hat{z}_b^\ell, \mathcal{Z}_k \rangle$
  Retrieve quantized embeddings: $z_{q,b}^\ell \leftarrow \mathcal{Z}_{a_{b,\ell}}$
  Reconstruct: $\hat{\mathcal{X}}_{t,b} \leftarrow \mathcal{G}_\phi(z_{q,b})$
  Compute the minibatch reconstruction and commitment loss using Eq. equation 8
  Update ψ, ϕ using straight-through gradients through quantization
  Update $\mathcal{Z}$ using EMA, reset inactive entries, and normalize codebook entries
end while
Freeze $\mathcal{E}_\psi$, $\mathcal{G}_\phi$, and $\mathcal{Z}$
return $\mathcal{E}_\psi, \mathcal{G}_\phi, \mathcal{Z}$

Visualization of Forecasting Results. Figure 6 presents forecasting examples from five datasets with varying dimensionalities. In these visualizations, ProtoFlow generally follows the main temporal variations in both relatively smooth and more fluctuating series, illustrating its ability to accommodate different temporal patterns.

Algorithm 2 ProtoFlow Stage 2: Codebook-Prior Flow Matching

Input: Training data $\mathcal{D} = \{(\mathcal{X}_p, \mathcal{X}_t, \mathbf{y}^c)\}$, frozen encoder $\mathcal{E}_\psi$ and codebook $\mathcal{Z}$
Parameters: Local rank r, perturbation scale γ, flow-time distribution q_t
Output: Vector field $\mathbf{v}_\theta$, context encoder $\mathcal{H}_\omega$, prior parameters $\mathcal{P}$
// Construct the source prior
Collect normalized encoder outputs and code assignments from training targets
Estimate $\mathcal{P} = \{(\pi_k, \mu_k, B_k, \sigma_k)\}_{k=1}^K$ from $\mathcal{Z}$ and the collected statistics
Fix $\mathcal{P}$ and initialize $\mathbf{v}_\theta, \mathcal{H}_\omega$
// Train conditional latent transport
while training is not complete **do**
  Sample a minibatch $\{(\mathcal{X}_{p,b}, \mathcal{X}_{t,b}, \mathbf{y}_b^c)\}_{b\in\mathcal{B}}$
  For each b, obtain the frozen target: $\mathbf{x}_{s,b}^1 \leftarrow q(\mathcal{E}_\psi(\mathcal{X}_{t,b}))$
  Construct conditions: $\mathbf{c}_b \leftarrow (\mathcal{H}_\omega(\mathcal{X}_{p,b}), \mathbf{y}_b^c)$
  Independently sample $\mathbf{x}_{s,b}^0 \sim P_{\mathcal{Z}}$ using Eqs. equation 11 and equation 12, and sample $t_b \sim q_t$
  Interpolate: $\mathbf{x}_{s,b}^{t_b} \leftarrow (1-t_b)\mathbf{x}_{s,b}^0 + t_b\mathbf{x}_{s,b}^1$
  Compute the velocity regression loss:

$$\mathcal{L}_{\mathrm{FM}} \leftarrow \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}}\left\|\mathbf{v}_\theta(\mathbf{x}_{s,b}^{t_b}, t_b \mid \mathbf{c}_b) - (\mathbf{x}_{s,b}^1 - \mathbf{x}_{s,b}^0)\right\|_F^2$$

  Update θ, ω by minimizing $\mathcal{L}_{\mathrm{FM}}$
end while
return $\mathbf{v}_\theta, \mathcal{H}_\omega, \mathcal{P}$

Algorithm 3 ProtoFlow Inference

Input: Observed history $\mathcal{X}_p$, future temporal features $\mathbf{y}^c$, prediction length τ
Trained components: Decoder $\mathcal{G}_\phi$, context encoder $\mathcal{H}_\omega$, vector field $\mathbf{v}_\theta$, prior parameters $\mathcal{P} = \{(\pi_k, \mu_k, B_k, \sigma_k)\}_{k=1}^K$
Parameters: Number of forecast samples N, downsampling ratio ρ, local rank r, perturbation scale γ, integration grid $0 = t_0 < \cdots < t_M = 1$
Output: Forecast samples $\{\hat{\mathcal{X}}_t^{(n)}\}_{n=1}^N$

// Encode forecasting conditions
$L \leftarrow \lfloor \tau/\rho \rfloor$
$\mathbf{c} \leftarrow (\mathcal{H}_\omega(\mathcal{X}_p), \mathbf{y}^c)$
for $n = 1$ to N **do**

  // Sample the source latent sequence
  Independently sample $s_\ell^0 \sim \mathrm{Cat}(\pi_1, \ldots, \pi_K)$ and $\varepsilon_\ell \sim \mathcal{N}(0, I_r)$ for $\ell = 1, \ldots, L$
  $\mathbf{x}_{s,\ell}^0 \leftarrow \mathrm{Norm}\Big(\mu_{s_\ell^0} + \gamma B_{s_\ell^0}(\sigma_{s_\ell^0} \odot \varepsilon_\ell)\Big)$
  $\hat{\mathbf{x}}_s^{t_0} \leftarrow \mathrm{StackRows}(\mathbf{x}_{s,1}^0, \ldots, \mathbf{x}_{s,L}^0)$

  // Joint latent transport
  for $j = 0$ to $M-1$ **do**
    $\Delta t_j \leftarrow t_{j+1} - t_j$
    $\hat{\mathbf{x}}_s^{t_{j+1}} \leftarrow \hat{\mathbf{x}}_s^{t_j} + \Delta t_j\, \mathbf{v}_\theta(\hat{\mathbf{x}}_s^{t_j}, t_j \mid \mathbf{c})$
  end for

  // Decode the forecast
  $\hat{\mathcal{X}}_t^{(n)} \leftarrow \mathcal{G}_\phi(\hat{\mathbf{x}}_s^{t_M})$
end for
return $\{\hat{\mathcal{X}}_t^{(n)}\}_{n=1}^N$

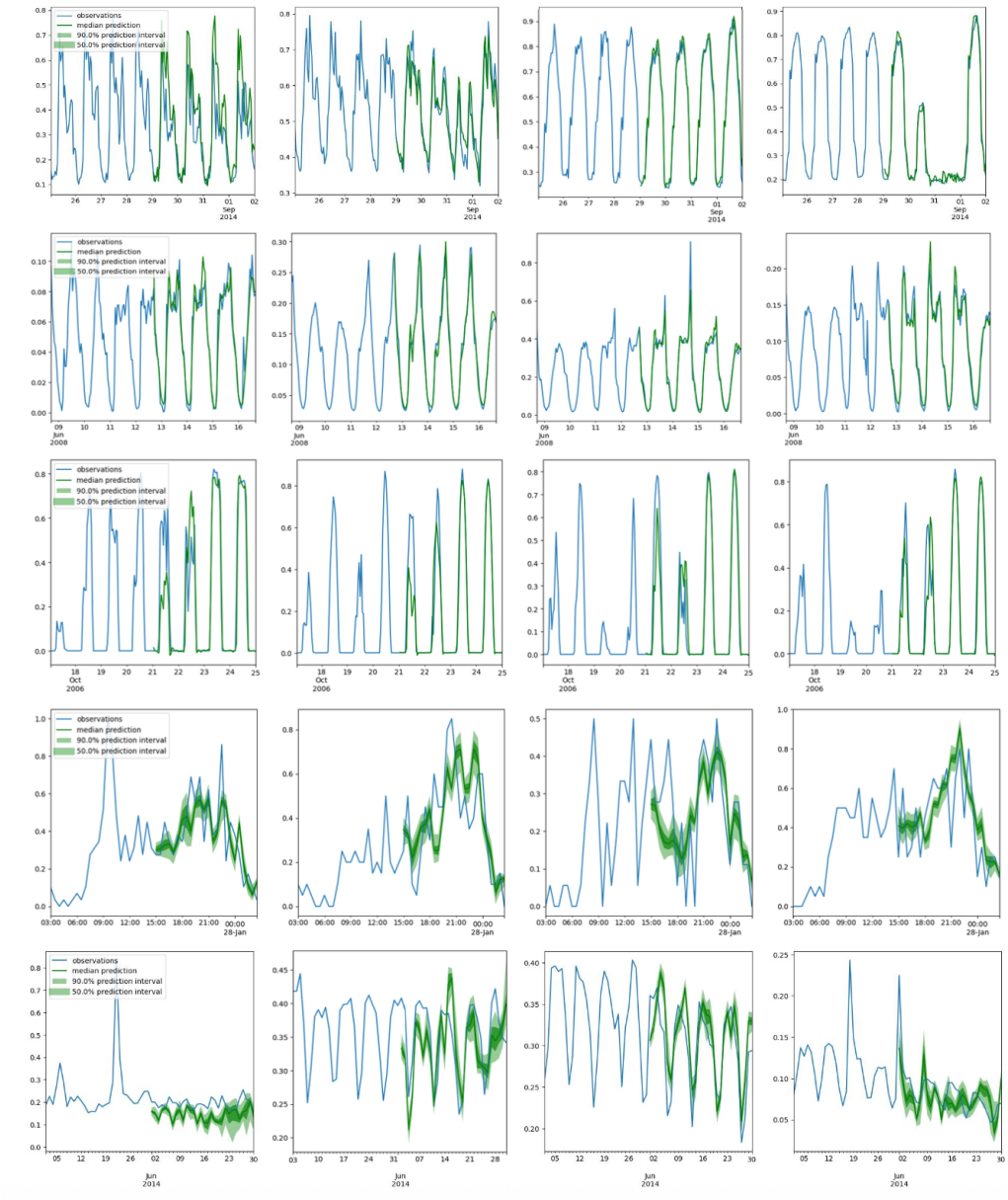

Figure 6: ProtoFlow forecasts with a prediction horizon of 96. Rows show Electricity (370 dimensions), Traffic (963 dimensions), Solar (137 dimensions), Taxi (1214 dimensions), and Wikipedia (2000 dimensions) from top to bottom, with four examples per dataset.

D LIMITATIONS

ProtoFlow constructs its prototype prior from global training statistics and keeps it fixed during forecasting. Although the learned transport is conditioned on historical observations, the source prior does not adapt its probability allocation to individual forecasting contexts. A promising direction is to develop context-dependent prototype weighting, allowing initialization to reflect the current temporal regime while retaining the efficiency of few-step generation. We leave this direction for our future work.