Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations

Anonymous authors
Paper under double-blind review

Abstract

We introduce a novel modeling approach for time series imputation and forecasting, tailored to address the challenges often encountered in real-world data, such as irregular samples, missing data, or unaligned measurements from multiple sensors. Our method relies on a continuous-time-dependent model of the series' evolution dynamics. It leverages adaptations of conditional, implicit neural representations for sequential data. A modulation mechanism, driven by a meta-learning algorithm, allows adaptation to unseen samples and extrapolation beyond observed time-windows for long-term predictions. The model provides a highly flexible and unified framework for imputation and forecasting tasks across a wide range of challenging scenarios. It achieves state-of-the-art performance on classical benchmarks and outperforms alternative time-continuous models.

1 Introduction

Time series analysis and modeling are ubiquitous in a wide range of fields, including industry, medicine, and climate science. The variety, heterogeneity and increasing number of deployed sensors, raise new challenges when dealing with real-world problems for which current methods often fail. For example, data are frequently irregularly sampled, contain missing values, or are unaligned when collected from distributed sensors (Schulz and Stattegger, 1997; Clark and Bjørnstad, 2004). Recent advancements in deep learning have significantly improved state-of-the-art performance in both data imputation (Cao et al., 2018; Du et al., 2023) and forecasting tasks (Zeng et al., 2022; Nie et al., 2022). Many state-of-the-art models, such as transformers, have been primarily designed for dense and regular grids (Wu et al., 2021; Nie et al., 2022; Du et al., 2023). They struggle to handle irregular data and often suffer from significant performance degradation (Chen et al., 2001; Kim et al., 2019).

Our objective is to explore alternatives to state-of-the-art (SOTA) transformers able to handle, in a unified framework, imputation and forecasting tasks for irregularly, arbitrarily sampled, and unaligned time series sources. Time-dependent continuous models (Rasmussen and Williams, 2006; Garnelo et al., 2018; Rubanova et al., 2019) offer such an alternative. However, until now, their performance has lagged significantly behind that of models designed for regular discrete grids. A few years ago, implicit neural representations (INRs) emerged as a powerful tool for representing images as continuous functions of spatial coordinates (Sitzmann et al., 2020; Tancik et al., 2020) with recent new applications such as image generation (Dupont et al., 2022) or even modeling dynamical systems (Yin et al., 2023).

In this work, we leverage the potential of conditional INR models within a meta-learning approach to introduce TimeFlow: a unified framework designed for modeling continuous time series and addressing imputation and forecasting tasks with irregular and unaligned observations. Our key contributions are:

- We propose a novel framework that excels in modeling time series as continuous functions of time, accepting arbitrary time step inputs, thus enabling the handling of irregular and unaligned time series for both imputation and forecasting tasks. This is one of the very first attempts to adapt INRs that enables efficient handling of both imputation and forecasting tasks within a unified framework. The methodology which leverages the synergy between the model components, evidenced in the context of this application, is a pioneering contribution to the field.

- We conducted an extensive comparison with state-of-the-art continuous and discrete models. It demonstrates that our approach outperforms continuous and discrete SOTA deep learning approaches for imputation. As for long-term forecasting, it outperforms existing continuous models both on regular and irregular samples. It is on par with SOTA discrete models on regularly sampled time series while allowing for a much greater flexibility for irregular samplings, allowing to cope with situations where discrete models fail. Furthermore, we prove that our method effortlessly handles previously unseen time series and new time windows, making it well-suited for real-world applications.

2 Related work

Discrete methods for time series imputation and forecasting. Recently, Deep Learning (DL) methods have been widely used for both time series imputation and forecasting. For imputation, BRITS (Cao et al., 2018) uses a bidirectional recurrent neural network (RNN). Alternative frameworks were later explored, e.g., GAN-based (Luo et al., 2018; 2019; Liu et al., 2019), VAE-based (Fortuin et al., 2020), diffusion-based (Tashiro et al., 2021), matrix factorization-based (TIDER, Liu et al., 2023) and transformer-based (SAITS, Du et al., 2023) approaches. These methods cannot handle irregular time series. In situations involving multiple sensors, such as those placed at different locations, incorporating new sensors necessitates retraining the entire model, thereby limiting their usability. For forecasting, most recent DL SOTA models are based on transformers. Initial approaches apply plain transformers directly to the series, each token being a series element (Zhou et al., 2021; Liu et al., 2022; Wu et al., 2021; Zhou et al., 2022). These transformers may underperform linear models as shown in (Zeng et al., 2022). PatchTST (Nie et al., 2022) significantly improved transformers SOTA performance by considering sub-series as tokens of the series. However, all these models cannot handle properly irregularly sampled look-back windows.

Continuous methods for time series. Gaussian Processes (Rasmussen and Williams, 2006) have been a popular family of methods for modeling time series as continuous functions. They require choosing an appropriate kernel (Corani et al., 2021) and may suffer limitations in large dimensions settings. Neural Processes (NPs) (Garnelo et al., 2018; Kim et al., 2019) parameterize Gaussian processes through an encoder-decoder architecture leading to more computationally efficient implementations. NPs have been used to model simple signals for imputation and forecasting tasks, but struggle with more complex signals. Bilos et al. (2023) parameterizes a Gaussian Process through a diffusion model, but the model has difficulty adapting to a large number of timestamps. Other approaches such as Brouwer et al. (2019) and Rubanova et al. (2019) model time series continuously with latent ordinary differential equations. mTAN (Shukla and Marlin, 2021), a transformer model, uses an attention mechanism to impute irregular time series. While these approaches have shown significant progress in continuous modeling for time series, we observed that their performances on imputation and forecasting tasks are inferior compared to the aforementioned discrete models (Table 1, Table 2).

Implicit neural representations. The recent development of implicit neural representations (INRs) has led to impressive results in computer vision (Sitzmann et al., 2020; Tancik et al., 2020; Fathony et al., 2021; Mildenhall et al., 2021). INRs can represent data as a continuous function, which can be queried at any coordinate. While they have been applied in other fields such as physics (Yin et al., 2023) and meteorology (Huang and Hoefler, 2023), there has been limited research on INRs for time series analysis. Prior works (Fons et al., 2022; Jeong and Shin, 2022) focused on time series generation for data augmentation and on time series encoding for reconstruction but are limited by their fixed grid input requirement. DeepTime (Woo et al., 2022) is the closest work to our contribution. DeepTime learns a set of basis INR functions from a training set of multiple time series and combines them using a Ridge regressor. This regressor allows it to adapt to new time series. It has been designed for forecasting only. The original version cannot handle imputation properly and was adapted to do so for our comparisons. In our experiments, we will demonstrate that TimeFlow significantly outperforms DeepTime in imputation and also in forecasting tasks when dealing with missing values in the look-back window. TimeFlow also shows a slight advantage over DeepTime in forecasting regularly sampled series.

3 The TimeFlow framework

3.1 Problem setting

We aim to develop a unified framework for time series imputation and forecasting that reduces dependency on a fixed sampling scheme for time series. We introduce the following notations for both tasks. During training, in the imputation setting, we have access to time series in an observed temporal grid denoted as $\mathcal{T}_{in}$, which is a subset of the dense temporal support $\mathcal{T}$. In the forecasting setting, we observe time series within a limited past time grid, referred to as the 'look-back window' and denoted as $\mathcal{T}_{in}$ (a subset of $\mathcal{T}$), as well as a future grid, the 'horizon', denoted as $\mathcal{T}_{out}$ (also a subset of $\mathcal{T}$). At test time, in both cases, and given observed values in a temporal grid $\mathcal{T}^*_{in}$ included in a possibly new temporal window $\mathcal{T}^*$, our objective is to infer the time series values within $\mathcal{T}^*$. $\mathcal{T}^* = \mathcal{T}$ if we infer values in the training temporal support (e.g. in the classical imputation scenario, see Section 4.1), $\mathcal{T}^* \neq \mathcal{T}$ if we infer for a new temporal support(e.g. in the forecasting setting, see Section 4.2).

3.2 Key components

Our framework is articulated around three key components:

(i) **INR-based time-continuous functions**: a discrete time series $x = (x_{t_1}, x_{t_2}, \ldots, x_{t_k})$ can be represented by an underlying time-continuous function $\mathbf{x}\colon t \in \mathbb{R}_+ \to x_t \in \mathbb{R}^d$ (in our experiments $d = 1$). We want to approximate the ground-truth $\mathbf{x}$ by employing implicit neural representations (INRs), which are neural networks capable of learning a parameterized continuous function f_θ from discrete data by minimizing the reconstruction loss between observed data and network's outputs.

(ii) **Conditional INRs with modulations**: An INR can represent only one function, whether it's an image or a time series. To effectively represent a collection of time series $(x^{(j)})_j$ using INRs, we improve their encoding by incorporating per-sample modulations, which we denote as $\psi^{(j)}$. These modulations condition the parameters θ of the INRs. We use the notation $f_{\theta,\psi^{(j)}}$ to refer to the conditioned INR with the modulations $\psi^{(j)}$.

(iii) **Optimization-based encoding**: the conditioning modulation parameters $\psi^{(j)}$ are calculated as a function of codes $z^{(j)}$ that represent the individual sample series. We acquire these codes $z^{(j)}$ through a meta-learning optimization process using an auto-decoding strategy. Notably, auto-decoding has been found to be more efficient for this purpose than set encoders (Kim et al., 2019).

In the following sections, we will elaborate on each component of our method. Given that the choices made for each component and the methodology developed to enhance their synergy are essential aspects, we provide a discussion of the various choices involved in Section 3.4.

INR-based time-continuous functions. We implement our INR with Fourier features and a feed-forward network (FFN) with ReLU activations, i.e. for a time coordinate $t \in \mathcal{T}$, the output of the INR f_θ is given by $f_\theta(t) = \text{FFN}(\gamma(t))$. The Fourier Features $\gamma(\cdot)$ are a frequency embedding of the time coordinates used to capture high-frequencies (Tancik et al., 2020; Mildenhall et al., 2021). In our case, we chose $\gamma(t) := (\sin(\pi t), \cos(\pi t), \cdots, \sin(2^{N-1}\pi t), \cos(2^{N-1}\pi t))$, with N the number of fixed frequencies. For an INR with L layers, the output is computed as follows: (i) we get the frequency embedding $\phi_0 = \gamma(t)$, (ii) we update the hidden states according to $\phi_l = \text{ReLU}(\theta_l \phi_{l-1} + b_l)$ for $l = 1, \ldots, L$, (iii) we project onto the output space $f_\theta(t) = \theta_{L+1}\phi_L + b_{L+1}$.

Conditional INRs with modulations. As indicated, sample conditioning of the INR is performed through modulations of its parameters. In order to adapt rapidly the model to new samples, the conditioning should rely only on a small number of the INR parameters. This is achieved by modifying only the biases of the INR through the introduction of an additional bias term $\psi_l^{(j)}$ for each layer l, also known as *shift modulation*. To further limit the versatility of the conditioning, we generate the instance modulations $\psi^{(j)}$ from compact codes $z^{(j)}$ through a linear hypernetwork h with parameters w, i.e., $\psi^{(j)} = h_w(z^{(j)})$.

Consequently, the approximation of a time series $x^{(j)}$, denoted globally as $f_{\theta,h_w(z^{(j)})}$, will depend on shared parameters θ and w that are common among all the INRs involved in modeling the series family and on the code $z^{(j)}$ specific to series $x^{(j)}$. The output of the l-th layer of the modulated INR is given by $\phi_l = \text{ReLU}(\theta_l \phi_{l-1} + b_l + \psi_l^{(j)})$, where $\psi_l^{(j)} = W_l z^{(j)}$, and $w := (W_l)_{l=1}^{L}$ are the parameters of the hypernetwork h_w. This design enables gathering information across samples into the common parameters of the INR and hypernetwork, while the codes contain only specific information about their respective time-series samples. The architecture is illustrated in Figure 1.

Optimization-based encoding. We condition the INR using the data from $\mathcal{T}_{in}$, and learn the shared INR and hypernetwork parameters θ and w using $\mathcal{T}_{in}$ for both imputation and forecasting, and $\mathcal{T}_{out}$ for forecasting only. We achieve the conditioning on $\mathcal{T}_{in}$ by optimizing the codes $z^{(j)}$ through gradient descent. The joint optimization of the codes and common parameters is challenging. In TimeFlow, it is achieved through a meta-learning approach, adapted from Dupont et al. (2022) and Zintgraf et al. (2019). The objective is to learn shared parameters so that the code $z^{(j)}$ can be adapted in just a few gradient steps for a new series $x^{(j)}$. For training, we perform parameter optimization at two levels: the inner-loop and the outer-loop. The inner-loop adapts the code $z^{(j)}$ to condition the network on the set $\mathcal{T}_{in}^{(j)}$, while the outer-loop updates the common parameters using $\mathcal{T}_{in}^{(j)}$ and also $\mathcal{T}_{out}^{(j)}$ for forecasting (see Appendix H for more detailed intuition.). We present our training optimization in Algorithm 1. At each training epoch and for each batch of data $\mathcal{B}$ composed of time series $x^{(j)}$ sampled from the training set, we first update individually the codes $z^{(j)}$ in the inner loop, before updating the common parameters in the outer loop using a loss over the whole batch. We introduce a parameter λ to weight the importance of the loss over $\mathcal{T}_{out}$ w.r.t. the loss over $\mathcal{T}_{in}$ for the outer-loop. In practice, when $\mathcal{T}_{out}$ exists, i.e. for forecasting, we set $\lambda = 1$ and $\lambda = 0$ otherwise. We use an MSE loss over the observations grid $\mathcal{L}_{\mathcal{T}}(x_t, \tilde{x}_t) := \mathbb{E}_{t\sim\mathcal{T}}[(x_t - \tilde{x}_t)^2]$. We denote α and η the learning rates of the inner-loop and outer-loop. Using $K = 3$ steps for training and testing is sufficient for our experiments thanks to the use of second-order meta-learning as explained in Section 3.4.

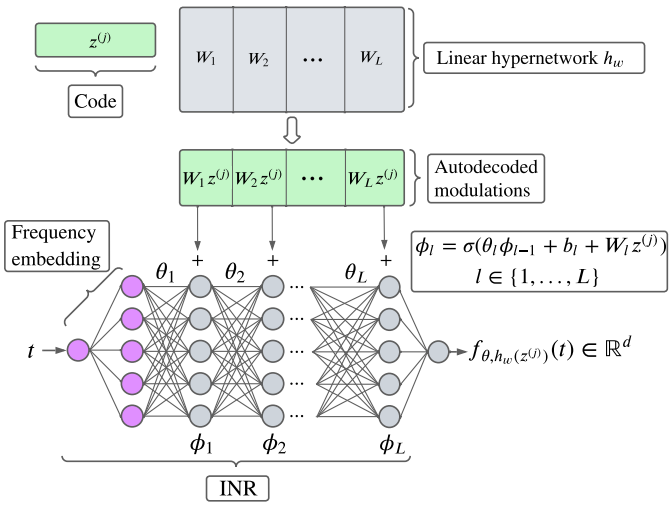


Figure 1: Overview of TimeFlow architecture. Forward pass to approximate the time series $x^{(j)}$. σ stands for the ReLU activation function.

Algorithm 1: TimeFlow Training

while *no convergence* **do**
- Sample batch $\mathcal{B}$ of data $(x^{(j)})_{j\in\mathcal{B}}$;
- Set codes to zero $z^{(j)} \leftarrow 0, \forall j \in \mathcal{B}$;
- // inner loop for encoding:
- **for** $j \in \mathcal{B}$ *and step* $\in \{1, ..., K\}$ **do**
  - $z^{(j)} \leftarrow z^{(j)} - \alpha \nabla_{z^{(j)}} \mathcal{L}_{\mathcal{T}_{in}^{(j)}}(f_{\theta,h_w(z^{(j)})}, x^{(j)})$;
- // outer loop step:
- $[\theta, w] \leftarrow [\theta, w] - \eta \nabla_{[\theta,w]} \frac{1}{|\mathcal{B}|} \sum_{j\in\mathcal{B}} [\mathcal{L}_{\mathcal{T}_{in}^{(j)}}(f_{\theta,h_w(z^{(j)})}, x^{(j)}) + \lambda \mathcal{L}_{\mathcal{T}_{out}^{(j)}}(f_{\theta,h_w(z^{(j)})}, x^{(j)})]$;

3.3 TimeFlow inference

During the inference process, we aim to infer the time series value for each timestamp in the dense grid $\mathcal{T}^{*(j)}$ based on the partial observation grid $\mathcal{T}_{in}^{*(j)} \subset \mathcal{T}^{*(j)}$. We can encounter two scenarios: (i) One where we

observe the same time window as during training ($\mathcal{T}^{*(j)} = \mathcal{T}^{(j)}$) as in the imputation setting in Section 4.1. (ii) One, where we are dealing with a newly observed time window ($\mathcal{T}^{*(j)} \neq \mathcal{T}^{(j)}$), as in the forecasting setting in Section 4.2. At inference, the parameters θ and w are kept fixed to their final training values. We optimize the individual parameters $z^{*(j)}$ based on the newly observed grid $\mathcal{T}_{in}^{*(j)}$ using the K inner-steps of the meta-learning algorithm as described in Algorithm 2. We are then in position to query $f_{\theta,h_w(z^{*(j)})}(t)$ for any given timestamp $t \in \mathcal{T}^{*(j)}$.

Algorithm 2: TimeFlow Inference with trained θ, w

For the j-th series ($x^{(j)}$), set code to zero $z^{*(j)} \leftarrow 0$;
for *step* $\in \{1, ..., K\}$ **do**
  $z^{*(j)} \leftarrow z^{*(j)} - \alpha \nabla_{z^{*(j)}} \mathcal{L}_{\mathcal{T}_{in}^{*(j)}}(f_{\theta,h_w(z^{*(j)})}, x_t)$
Query $f_{\theta,h_w(z^{*(j)})}(t)$ for any $t \in \mathcal{T}^{*(j)}$

3.4 Discussion on implementation choices

As indicated before, adapting the components and enhancing their synergy for the tasks of imputation and forecasting is not trivial and requires careful choices. We conducted several ablation studies to provide a comprehensive examination of key implementation choices of our framework.

Our findings can be summarized as follows.

- **Choice of INR**: An FFN with Fourier Features outperformed other popular INRs for the tasks considered in this study. Unlike SIREN (Sitzmann et al., 2020), which does not explicitly incorporate frequencies but uses sine activation functions, the Fourier features network can more effectively capture a wider range of frequencies, especially at low sampling rates. This is crucial for accurately capturing high frequencies in sparsely observed time series. Our experiments, detailed in Section B.2.1 and Table 4, demonstrate this superiority across various datasets.
- **Choice of encoding / meta-learning**: TimeFlow with a set encoder for learning the compact conditioning codes z in place of the auto-decoding strategy used here, proved much less effective on complex datasets. This is further elaborated in Section B.2.4 and Table 11. Additionally, replacing the 2nd-order meta-learning optimization for a 1st-order one, such as REPTILE (Nichol et al., 2018), led to unstable training, as shown in Table 10.
- **Choice of modulations**: Complexifying the modulation by introducing scaling parameters in addition to shift parameters did not provide performance gains. Our experiments on the *Electricity* dataset, detailed in Section B.2.5 and Table 12, indicate that shift-only modulation is more efficient.

For TimeFlow, across all experiments, we used a code dimension of 128, an FFN with a depth of 5 and a width of 256, and 64 Fourier features. We used 3 inner steps and a learning rate of 0.01 for the inner-loop, and a learning rate of 5×10^{-4} for the outer-loop. We performed a comprehensive analysis to understand notably the **influence of the z dimension**: a latent code dimension of 128 was suitable for our tasks; this is supported by results in Section B.2.2 and Table 5 - and the **influence of the number of inner steps**: using 3 inner steps for training and inference struck a favorable balance between reconstruction capabilities and computational efficiency, as detailed in Section B.2.3.

4 Experiments

We conducted a comprehensive evaluation of our TimeFlow framework across three different tasks, comparing its performance to state-of-the-art continuous and discrete baseline methods. In Section 4.1, we assess TimeFlow's capabilities to impute sparsely observed time series under various sampling rates. Section 4.2 focuses on long-term forecasting, where we evaluate TimeFlow over standard long-term forecasting horizons. In Section 4.3, we tackle a challenging task forecasting with incomplete look-back windows, thus combining the challenges of imputation and forecasting. The code for the experiments is available at this link

Datasets. We tested our framework on three extensive multivariate datasets where a single phenomenon is measured at multiple locations over time, namely *Electricity*, *Traffic* and *Solar*. The *Electricity* dataset comprises hourly electricity load curves of 321 customers in Portugal, spanning the years 2012 to 2014. The *Traffic* dataset is composed of hourly road occupancy rates from 862 locations in San Francisco during 2015 and 2016. Lastly, the *Solar* dataset contains measurements of solar power production from 137 photovoltaic plants in Alabama, recorded at 10-minute intervals in 2006. Additionally, we have created an hourly version, *SolarH*, for the sake of consistency in the forecasting section. These datasets exhibit diversity in various characteristics: • They exhibit diverse temporal frequencies, including daily and weekly seasonality observed in the *Traffic* and *Electricity* datasets, while the *Solar* dataset possesses only daily frequency. • There is individual variability across data samples and more pronounced trends in the *Electricity* dataset compared to the *Traffic* and *Solar* datasets.

4.1 Imputation

We consider the classical imputation setting where n time series are partially observed over a given time window. Using our approach, we can predict for each time series the value at any timestamp t in that time window based on partial observations.

Setting. For a time series $x^{(j)}$, we denote the set of observed points as $\mathcal{T}_{in}^{(j)}$ and the ground truth set of points as $\mathcal{T}^{(j)}$. The observed time grids may be irregularly spaced and may differ across the different time series ($\mathcal{T}_{in}^{(j_1)} \neq \mathcal{T}_{in}^{(j_2)}, \forall j_1 \neq j_2$). The model is trained for each $x^{(j)}$ following Algorithm 1. Then, we aim to infer for any unobserved $t \in \mathcal{T}^{(j)}$ the missing value $x_t^{(j)}$ conditioned on $\mathcal{T}_{in}^{(j)}$ according to Algorithm 2. For this imputation task, the TimeFlow training and inference procedures are detailed in Section 3 and illustrated in Figure 2. For comparison with the SOTA imputation baselines, we assume that the ground truth time grid is the same for each sample. The subsampling rate τ is define as the rate of observed values.

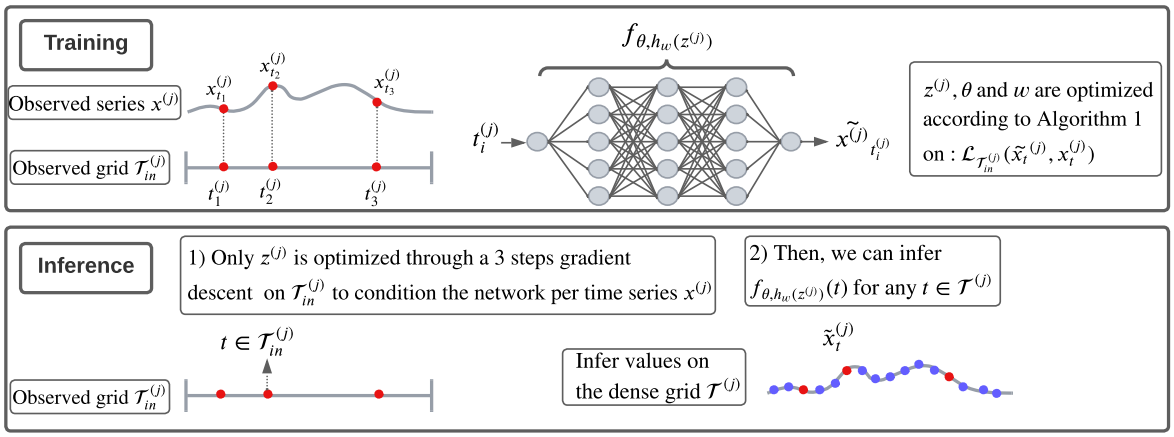


Figure 2: Training and inference procedures of TimeFlow for imputation. (i) During training, for each time series $x^{(j)}$, our observations (red dots •) are restricted to the sparsely sampled grid, denoted as $\mathcal{T}_{in}^{(j)}$. (ii) During inference, our objective is to infer the values over the dense grids $\mathcal{T}^{(j)}$, on the unobserved data points (such as the blue dots • on the figure).

Baselines. We compare TimeFlow with various baselines, including discrete imputation methods, such as CSDI (Tashiro et al., 2021), SAITS (Du et al., 2023), BRITS (Cao et al., 2018), and TIDER (Liu et al., 2023), and continuous ones, such as Neural Process (NP, Garnelo et al., 2018), mTAN (Shukla and Marlin, 2021), and DeepTime with slight adjustments (Woo et al., 2022) (details cf. Appendix D.3). See Appendix D.1.1 for the baseline training procedure and hyperparameter selection. For each dataset, we divide the series into five independent time windows (consisting of 2000 timestamps for *Electricity* and *Traffic*, and 10,000 timestamps for *Solar*), perform imputation on each time window and average the performance to obtain robust results.

Table 1: Mean MAE imputation results on the missing grid only. Each time series is divided into 5 time windows onto which imputation is performed, and the performances are averaged over the 5 windows. In the table, τ stands for the subsampling rate, i.e. the proportion of observed points considered for each time window. Bold results are best, underlined results are second best. TimeFlow improvement represents the overall percentage improvement achieved by TimeFlow compared to the specific method being considered.

		Continuous methods				Discrete methods			
	τ	TimeFlow	DeepTime	mTAN	Neural Process	CSDI	SAITS	BRITS	TIDER
Electricity	0.05	**0.324 ± 0.013**	0.379 ± 0.037	0.575 ± 0.039	0.357 ± 0.015	0.462 ± 0.021	0.384 ± 0.019	0.329 ± 0.015	0.427 ± 0.010
	0.10	**0.250 ± 0.010**	0.333 ± 0.034	0.412 ± 0.047	0.417 ± 0.057	0.398 ± 0.072	0.308 ± 0.011	0.287 ± 0.015	0.399 ± 0.009
	0.20	**0.225 ± 0.008**	0.244 ± 0.013	0.342 ± 0.014	0.320 ± 0.017	0.341 ± 0.068	0.261 ± 0.008	0.245 ± 0.011	0.391 ± 0.010
	0.30	**0.212 ± 0.007**	0.240 ± 0.014	0.335 ± 0.015	0.300 ± 0.022	0.277 ± 0.059	0.236 ± 0.008	0.221 ± 0.008	0.384 ± 0.009
	0.50	0.194 ± 0.007	0.227 ± 0.012	0.340 ± 0.022	0.297 ± 0.016	**0.168 ± 0.003**	0.209 ± 0.008	0.193 ± 0.008	0.386 ± 0.009
Solar	0.05	**0.095 ± 0.015**	0.190 ± 0.020	0.241 ± 0.102	0.115 ± 0.015	0.374 ± 0.033	0.142 ± 0.016	0.165 ± 0.014	0.291 ± 0.009
	0.10	**0.083 ± 0.015**	0.159 ± 0.013	0.251 ± 0.081	0.114 ± 0.014	0.375 ± 0.038	0.124 ± 0.018	0.132 ± 0.015	0.276 ± 0.010
	0.20	**0.072 ± 0.015**	0.149 ± 0.020	0.314 ± 0.035	0.109 ± 0.016	0.217 ± 0.023	0.108 ± 0.014	0.109 ± 0.012	0.270 ± 0.010
	0.30	**0.061 ± 0.012**	0.135 ± 0.014	0.338 ± 0.05	0.108 ± 0.016	0.156 ± 0.002	0.100 ± 0.015	0.098 ± 0.012	0.266 ± 0.010
	0.50	**0.054 ± 0.013**	0.098 ± 0.013	0.315 ± 0.080	0.107 ± 0.015	0.079 ± 0.011	0.094 ± 0.013	0.088 ± 0.013	0.262 ± 0.009
Traffic	0.05	0.283 ± 0.016	**0.246 ± 0.010**	0.406 ± 0.074	0.318 ± 0.014	0.337 ± 0.045	0.293 ± 0.007	0.261 ± 0.010	0.363 ± 0.007
	0.10	**0.211 ± 0.012**	0.214 ± 0.007	0.319 ± 0.025	0.288 ± 0.018	0.288 ± 0.017	0.237 ± 0.006	0.245 ± 0.009	0.362 ± 0.006
	0.20	**0.168 ± 0.006**	0.216 ± 0.006	0.270 ± 0.012	0.271 ± 0.011	0.269 ± 0.017	0.197 ± 0.005	0.224 ± 0.008	0.361 ± 0.006
	0.30	**0.151 ± 0.007**	0.172 ± 0.008	0.251 ± 0.006	0.259 ± 0.012	0.240 ± 0.037	0.180 ± 0.006	0.197 ± 0.007	0.355 ± 0.006
	0.50	**0.139 ± 0.007**	0.171 ± 0.005	0.278 ± 0.040	0.240 ± 0.021	0.144 ± 0.022	0.160 ± 0.008	0.161 ± 0.060	0.354 ± 0.007
TimeFlow improvement		/	24.14 %	50.53 %	31.61 %	36.12 %	20.33 %	18.90 %	53.40 %

We evaluate the quality of the models for different subsampling rates, from the easiest $\tau = 0.5$ to the most difficult $\tau = 0.05$. All the scores presented in the experiments are reported as Mean Absolute Error (MAE).

Results. We show in Table 1 that TimeFlow outperforms both discrete and continuous models across almost all τ's for the given datasets. The relative improvements of TimeFlow, as defined in Appendix C, over baselines are significant, ranging from 15% to 50%. Especially for the lowest sampling rate $\tau = 0.05$, TimeFlow outperforms all discrete baselines, demonstrating the advantages of continuous modeling. Additionally, it achieves lower imputation errors compared to continuous models in all but one cases. Qualitatively, we see on example series in Figure 3 that our model shows significant imputation capabilities, with a subsampling rate at $\tau = 0.1$ on the *Electricity* dataset. It captures well different frequencies and amplitudes in a challenging case (sample 35), although it underestimates the amplitude of some peaks. In a more challenging scenario (sample 25), where the series exhibit additional trend changes and frequency variations within the data, TimeFlow correctly imputes most timestamps, outperforming BRITS, which is the best-performing method for the *Electricity* dataset.

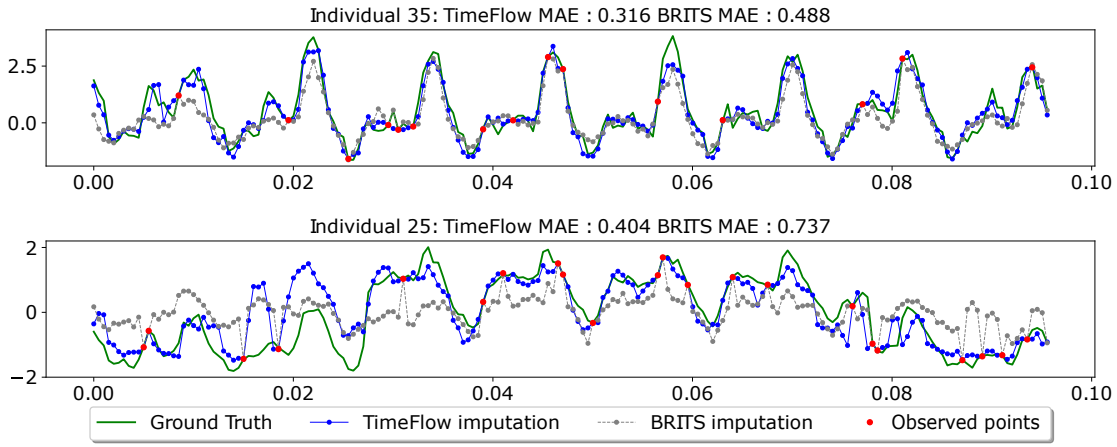


Figure 3: *Electricity dataset.* TimeFlow imputation (blue line) and BRITS imputation (gray line) with 10% of known point (red points) on the eight first days of samples 35 (top) and 25 (bottom).

Imputation on previously unseen time series. In more practical scenarios, such as cases involving the installation of new sensors, we often encounter new time series originating from the same underlying phenomenon. In such instances, it becomes crucial to make inferences for these previously unseen time series. Thanks to efficient adaptation in latent space, our model can easily be applied to these new time series (as shown in Appendix D.2, Table 18), contrasting with SOTA methods like SAITS and BRITS, which require full model retraining on the whole set of time series.

4.2 Forecasting

4.2.1 Forecasting for known time series

In this section, we are interested in the conventional long-term forecasting scenario. It consists in predicting the phenomenon in a specific future period, the horizon, based on the history of a limited past period, the look-back window. The forecaster is trained on a set of n observed time series for a given time window (train period) and tested on new distinct time windows.

Setting. For a given time series $x^{(j)}$, $\mathcal{T}_{in}^{(j)}$ denotes the look-back window and $\mathcal{T}_{out}^{(j)}$ the horizon of H points. During training, at each epoch, we train $f_{\theta,h_w(z^{(j)})}$ following Algorithm 1 with randomly drawn pairs of look-back window and horizon $(\mathcal{T}_{in}^{(j)} \cup \mathcal{T}_{out}^{(j)})_{j\in\mathcal{B}}$ within the observed train period. Then, for a distinct new time window $\mathcal{T}^{*(j)}$, given a look-back window $\mathcal{T}_{in}^{*(j)}$ we forecast future values any $t \in \mathcal{T}^{*(j)}$, the horizon interval, following Algorithm 2. We illustrate the training and inference of TimeFlow for the forecasting task in Figure 4. For further insight into the training window and inference periods, as well as additional experiments conducted under different inference scenarios, see Appendix E.1.

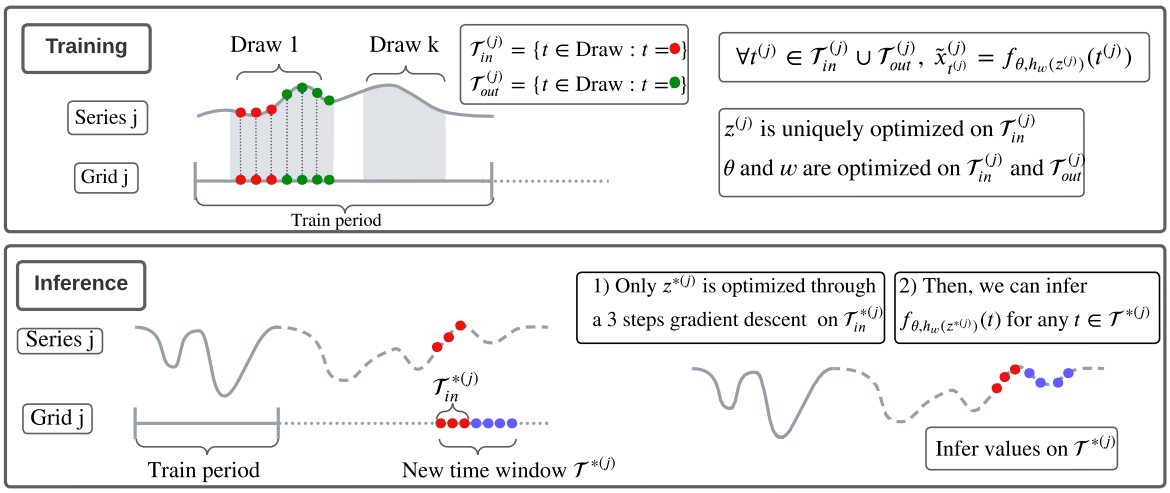


Figure 4: Training and inference procedure of TimeFlow for forecasting. (i) During training (top-figure), for each time series $x^{(j)}$, we observe some look-back window/horizon drawing pairs in the trained period. TimeFlow is trained with Algorithm 1 to predict all observed timestamps in this drawing pairs while being conditioned by the observed look-back window. (ii) Once TimeFlow is optimized, the objective during inference (bottom-figure) is to infer the horizon over new time windows (blue dots •) while being conditioned by the newly observed look-back window (red dots •).

Baselines. To evaluate the quality of our model in long-term forecasting, we compare it to the discrete baselines PatchTST (Nie et al., 2022), DLinear (Zeng et al., 2022), AutoFormer (Wu et al., 2021), and Informer (Zhou et al., 2021). We also include continuous baselines DeepTime and Neural Process (NP). See Appendix E.3.1 for the baseline training procedure and hyperparameter selection. In Table 2, we present the forecasting results for standard horizons in long-term forecasting: $H \in \{96, 192, 336, 720\}$. The look-back window length is fixed to 512.

Table 2: Mean MAE forecast results averaged over different time windows. Each time, the model is trained on one time window and tested on the others (there are 2 windows for *SolarH* and 5 for *Electricity* and *Traffic*). H stands for the horizon. Bold results are best, and underlined results are second best. TimeFlow improvement represents the overall percentage improvement achieved by TimeFlow compared to the specific method being considered.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.228 ± 0.028	0.244 ± 0.026	0.392 ± 0.045	**0.221 ± 0.023**	0.241 ± 0.030	0.546 ± 0.277	0.603 ± 0.255
	192	0.238 ± 0.020	0.252 ± 0.019	0.401 ± 0.046	**0.229 ± 0.020**	0.252 ± 0.025	0.500 ± 0.190	0.690 ± 0.291
	336	0.270 ± 0.031	0.284 ± 0.034	0.434 ± 0.076	**0.251 ± 0.027**	0.288 ± 0.038	0.523 ± 0.188	0.736 ± 0.271
	720	0.316 ± 0.055	0.359 ± 0.051	0.607 ± 0.150	**0.297 ± 0.039**	0.365 ± 0.059	0.631 ± 0.237	0.746 ± 0.265
SolarH	96	**0.190 ± 0.013**	**0.190 ± 0.020**	0.221 ± 0.048	0.262 ± 0.070	0.208 ± 0.014	0.245 ± 0.045	0.248 ± 0.022
	192	**0.202 ± 0.020**	0.204 ± 0.028	0.244 ± 0.048	0.253 ± 0.051	0.217 ± 0.022	0.333 ± 0.107	0.270 ± 0.031
	336	0.209 ± 0.017	**0.199 ± 0.026**	0.240 ± 0.006	0.259 ± 0.071	0.217 ± 0.026	0.334 ± 0.079	0.328 ± 0.048
	720	**0.218 ± 0.041**	0.229 ± 0.024	0.403 ± 0.147	0.267 ± 0.064	0.249 ± 0.034	0.351 ± 0.055	0.337 ± 0.037
Traffic	96	0.217 ± 0.032	0.228 ± 0.032	0.283 ± 0.027	**0.203 ± 0.037**	0.228 ± 0.033	0.319 ± 0.059	0.372 ± 0.078
	192	0.212 ± 0.028	0.220 ± 0.022	0.292 ± 0.024	**0.197 ± 0.030**	0.221 ± 0.023	0.368 ± 0.057	0.511 ± 0.247
	336	0.238 ± 0.034	0.245 ± 0.038	0.305 ± 0.039	**0.222 ± 0.039**	0.250 ± 0.040	0.434 ± 0.061	0.561 ± 0.263
	720	0.279 ± 0.050	0.290 ± 0.052	0.339 ± 0.038	**0.269 ± 0.057**	0.300 ± 0.057	0.462 ± 0.062	0.638 ± 0.067
TimeFlow improvement		/	3.74 %	29.06 %	3.23 %	6.92 %	42.09 %	48.57 %

Results. The results in Table 2 show that our approach ranks in the top two across all datasets and horizons and is the overall best continuous method. TimeFlow's performance is comparable to the current SOTA model PatchTST, with only 2% relative difference. Moreover, TimeFlow shows consistent results across the three datasets, whereas the other best discrete and continuous baselines, i.e. PatchTST and DeepTime, performance drops for some datasets. We also note that, despite the great performance of the SOTA PatchTST, other transformer-based baselines (discrete methods in Table 2) perform poorly. We provide a detailed insight on these results in Appendix E.1. Overall, although this evaluation setting favors discrete methods because the time series are observed at evenly distributed time steps, TimeFlow consistently performs as well as PatchTST and outperforms all the other methods, whether discrete or continuous. It is the first time that a continuous model has achieved the same level of performance as discrete methods within their specific setting.

4.2.2 Forecasting on previously unseen time series.

This section discusses how TimeFlow adapts to unseen time series, which is critical in forecasting. Indeed, in many real-world applications, forecasters are trained on a limited subset of available samples and applied to a wider range of samples during inference. Informer, AutoFormer, or DLinear original architectures directly model the relationships between time series (channel-dependence), limiting their adaptability to new samples. In contrast, TimeFlow takes a different approach by considering the observed series at different locations as distinct samples, similar to PatchTST, Neural Process, and DeepTime. This independence allows TimeFlow to effectively generalize to previously unseen time series of the same phenomenon.

Setting. In this setting, we propose to evaluate how TimeFlow performs on previously unseen time series. We compare it to the best forecaster, PatchTST. We train TimeFlow and PatchTST on 50 % of the samples and consider the remaining 50 % as the new time series. The training procedure is the same as described in Figure 4. In Figure 5, we present the results of TimeFlow and PatchTST for both known and new samples (for periods outside the training window).

Results. The results in Figure 5 highlight two key observations. First, both approaches show robust adaptability to new samples, as evidenced by the minimal difference in mean absolute error between known and new samples at inference. Second, TimeFlow and PatchTST exhibit comparable performance in this context, with negligible differences across horizons and datasets.

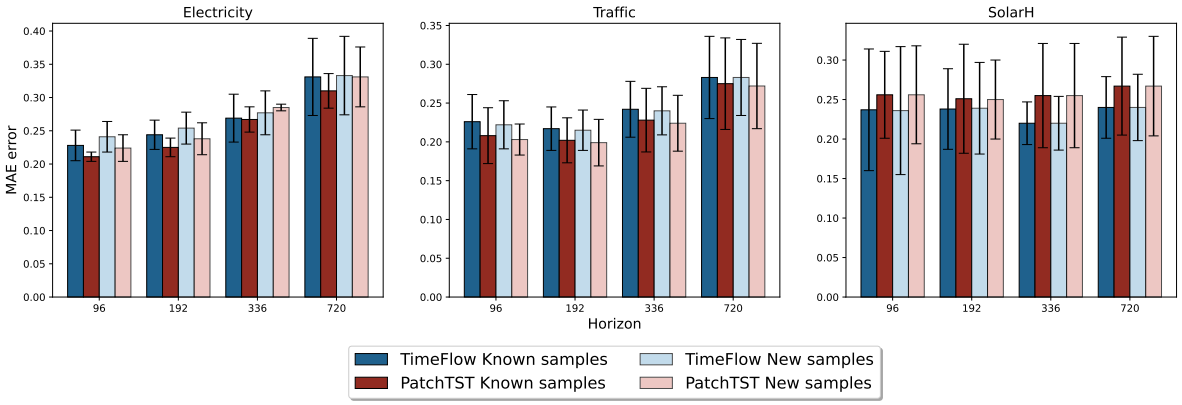


Figure 5: Mean MAE forecasting task results over different horizons in the context of generalization to new time series. Comparison of TimeFlow and PatchTST performances on the *Electricity*, *Traffic* and *SolarH* datasets.

4.3 Challenging task: Forecast while imputing incomplete look-back windows

In real-world scenarios, it is common to encounter missing or irregularly sampled series when making predictions on new time windows (Cinar et al., 2018; Tang et al., 2020). Continuous methods can handle these cases, as they are designed to accommodate irregular sampling within the look-back window. In this section, we formulate a task to simulate these real-world scenarios. It's worth noting that this task is often encountered in practice but is rarely considered in the DL literature.

Setting and baselines. This scenario is similar to the forecast setting in Section 4.2 and illustrated in Figure 4. The difference is that during inference, the look-back window is subsampled at a rate τ smaller than the one used for the training phase. This simulates a situation with missing observations in the look back window. Consequently, two distinct tasks emerge during the inference phase: imputing missing points within the sparsely observed look-back window, and forecasting over the horizon with this degraded context. In Table 3, we compare to the two other continuous baselines, DeepTime and NP on *Electricity* and *Traffic* for different τ's and horizons.

Table 3: MAE results for forecasting with missing values in the look-back window. τ stands for the percentage of observed values in the look-back window. Best results are in bold. TimeFlow improvement represents the overall percentage improvement (for each task) achieved by TimeFlow compared to the specific method being considered.

			TimeFlow		DeepTime		Neural Process	
	H	τ	Imputation error	Forecast error	Imputation error	Forecast error	Imputation error	Forecast error
Electricity	96	0.5	**0.151 ± 0.003**	**0.239 ± 0.013**	0.209 ± 0.004	0.270 ± 0.019	0.460 ± 0.048	0.486 ± 0.078
		0.2	**0.208 ± 0.006**	**0.260 ± 0.015**	0.249 ± 0.006	0.296 ± 0.023	0.644 ± 0.079	0.650 ± 0.095
		0.1	**0.272 ± 0.006**	**0.295 ± 0.016**	0.284 ± 0.007	0.324 ± 0.026	0.740 ± 0.083	0.737 ± 0.106
	192	0.5	**0.149 ± 0.004**	**0.235 ± 0.011**	0.204 ± 0.004	0.265 ± 0.018	0.461 ± 0.045	0.498 ± 0.070
		0.2	**0.209 ± 0.006**	**0.257 ± 0.013**	0.244 ± 0.007	0.290 ± 0.023	0.601 ± 0.075	0.626 ± 0.101
		0.1	**0.274 ± 0.010**	**0.289 ± 0.016**	0.282 ± 0.007	0.315 ± 0.025	0.461 ± 0.045	0.724 ± 0.090
Traffic	96	0.5	**0.180 ± 0.016**	**0.219 ± 0.026**	0.272 ± 0.028	0.243 ± 0.030	0.436 ± 0.025	0.444 ± 0.047
		0.2	**0.239 ± 0.019**	**0.243 ± 0.027**	0.335 ± 0.026	0.293 ± 0.027	0.596 ± 0.049	0.597 ± 0.075
		0.1	**0.312 ± 0.020**	**0.290 ± 0.027**	0.385 ± 0.025	0.344 ± 0.027	0.734 ± 0.102	0.731 ± 0.132
	192	0.5	**0.176 ± 0.014**	**0.217 ± 0.017**	0.241 ± 0.027	0.234 ± 0.021	0.477 ± 0.042	0.476 ± 0.043
		0.2	**0.233 ± 0.017**	**0.236 ± 0.021**	0.286 ± 0.027	0.276 ± 0.020	0.685 ± 0.109	0.678 ± 0.108
		0.1	**0.304 ± 0.019**	**0.277 ± 0.021**	0.331 ± 0.025	0.324 ± 0.021	0.888 ± 0.178	0.877 ± 0.174
TimeFlow improvement			/	/	18.97 %	11.87 %	61.88 %	58.41 %

Results. In Table 3, the results show that TimeFlow consistently outperforms other methods in imputation and forecasting for every scenarios. When comparing with the complete look-back windows observations scenario from Table 2, one observes that at a 0.5 sampling rate, TimeFlow presents only a slight reduction in performance, whereas other baseline methods experience more significant drops. For instance, when we compare forecast results between a complete window and a $\tau = 0.5$ subsampled window for *Electricity* with a forecasting horizon of $H = 96$, TimeFlow's error increases by a mere 4.6% (from 0.228 to 0.239). In contrast, DeepTime's error grows by over 10% (from 0.244 to 0.270), and NP experiences a rise of around 25% (from 0.392 to 0.486). For lower sampling rates, TimeFlow still delivers correct predictions. Qualitatively, we see on the series example in Figure 6 that despite observing only 10% of the look-back window, the model can correctly infer both the complete look-back window and the horizon. Both quantitative and qualitative results show the robustness and efficiency of TimeFlow on this particularly challenging setting.

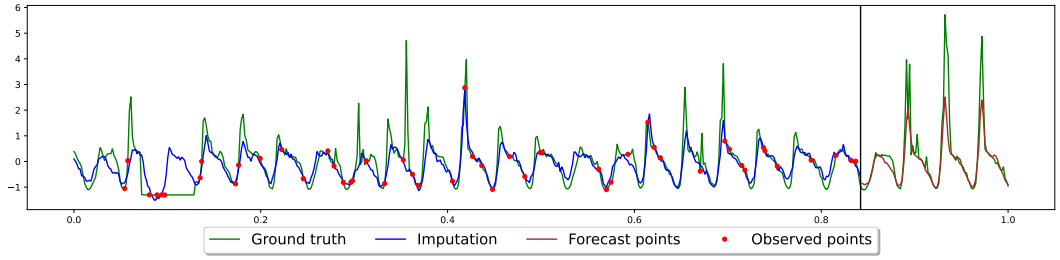


Figure 6: *Traffic dataset, sample 95.* In this figure, TimeFlow simultaneously imputes and forecasts at horizon 96 with a 10% partially observed look-back window of length 512.

5 Limitations

While TimeFlow shows promising performance across various tasks and settings, it is important to recognize some limitations. First, due to its auto-decoding process, TimeFlow tends to be significantly slower at inference time compared to other baselines by one to two orders of magnitude (Table 23). In addition, although TimeFlow effectively handles sets of homogeneous time series, additional mechanisms are required to handle heterogeneous time series with different frequencies effectively. The per-context shift modulation mechanism does not allow TimeFlow to fit time series with drastically different structures. This also explains why TimeFlow is particularly well adapted to regular frequency patterns. In the experiment section, all considered datasets exhibit pronounced periodicities. Finally, it should be noted that effective training of TimeFlow requires a relatively large number of samples (typically ≥ 100) to allow the model to accurately distinguish between individual patterns and shared information.

6 Conclusion

We have introduced a unified framework for continuous time series modeling leveraging conditional INR and meta-learning. Our experiments have demonstrated superior performance compared to other continuous methods, and better or comparable results to SOTA discrete methods. One of the standout features of our framework is its inherent continuity and the ability to modulate the INR parameters. This unique flexibility lets TimeFlow effectively tackle a wide array of challenges, including forecasting in the presence of missing values, accommodating irregular time steps, and extending the trained model's applicability to previously unseen time series and new time windows. Our empirical results have shown TimeFlow's effectiveness in handling homogeneous multivariate time series. As a logical next step, extending TimeFlow's capabilities to address heterogeneous multivariate phenomena represents a promising direction for future research.

References

M. Bilos, K. Rasul, A. Schneider, Y. Nevmyvaka, and S. Günnemann. Modeling temporal data as continuous functions with stochastic process diffusion. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato,

and J. Scarlett, editors, *International Conference on Machine Learning, ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 2452–2470. PMLR, 2023.

E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau. Gru-ode-bayes: continuous modeling of sporadically-observed time series. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 7379–7390, 2019.

W. Cao, D. Wang, J. Li, H. Zhou, Y. Li, and L. Li. Brits: bidirectional recurrent imputation for time series. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6776–6786, 2018.

H. Chen, S. Grant-Muller, L. Mussone, and F. Montgomery. A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. *Neural Computing & Applications*, 10:277–286, 2001.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Y. G. Cinar, H. Mirisaee, P. Goswami, É. Gaussier, and A. Aït-Bachir. Period-aware content attention rnns for time series forecasting with missing values. *Neurocomputing*, 312:177–186, 2018.

J. S. Clark and O. N. Bjørnstad. Population time series: process variability, observation errors, missing values, lags, and hidden states. *Ecology*, 85(11):3140–3150, 2004.

G. Corani, A. Benavoli, and M. Zaffalon. Time series forecasting with gaussian processes needs priors. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV 21*, pages 103–117. Springer, 2021.

W. Du, D. Côté, and Y. Liu. Saits: Self-attention-based imputation for time series. *Expert Systems with Applications*, 219:119619, 2023.

E. Dupont, H. Kim, S. M. A. Eslami, D. J. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5694–5725. PMLR, 2022.

R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

E. Fons, A. Sztrajman, Y. El-Laham, A. Iosifidis, and S. Vyetrenko. Hypertime: Implicit neural representation for time series. *CoRR*, abs/2208.05836, 2022.

V. Fortuin, D. Baranchuk, G. Rätsch, and S. Mandt. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*, pages 1651–1661. PMLR, 2020.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami. Conditional neural processes. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80, pages 1690–1699. PMLR, 2018.

T. J. Hastie. Generalized additive models. In *Statistical models in S*, pages 249–307. Routledge, 2017.

L. Huang and T. Hoefler. Compressing multidimensional weather and climate data into neural networks. In *International Conference on Learning Representations, ICLR*, 2023.

R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

K. Jeong and Y. Shin. Time-series anomaly detection with implicit neural representation. *CoRR*, abs/2201.11950, 2022.

T. Kim, W. Ko, and J. Kim. Analysis and impact evaluation of missing data imputation in day-ahead pv generation forecasting. *Applied Sciences*, 9(1):204, 2019.

S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

S. Liu, X. Li, G. Cong, Y. Chen, and Y. Jiang. Multivariate time-series imputation with disentangled temporal representations. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023.

Y. Liu, R. Yu, S. Zheng, E. Zhan, and Y. Yue. Naomi: Non-autoregressive multiresolution sequence imputation. *Advances in neural information processing systems*, 32, 2019.

Y. Luo, X. Cai, Y. Zhang, J. Xu, et al. Multivariate time series imputation with generative adversarial networks. *Advances in neural information processing systems*, 31, 2018.

Y. Luo, Y. Zhang, X. Cai, and X. Yuan. E2gan: End-to-end generative adversarial network for multivariate time series imputation. In *Proceedings of the 28th international joint conference on artificial intelligence*, pages 3094–3100. AAAI Press, 2019.

B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *CoRR*, abs/2211.14730, 2022.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019.

M. Schulz and K. Stattegger. Spectrum: Spectral analysis of unevenly spaced paleoclimatic time series. *Computers & Geosciences*, 23(9):929–945, 1997.

S. N. Shukla and B. M. Marlin. Multi-time attention networks for irregularly sampled time series. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

X. Tang, H. Yao, Y. Sun, C. C. Aggarwal, P. Mitra, and S. Wang. Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*, pages 5956–5963. AAAI Press, 2020.

Y. Tashiro, J. Song, Y. Song, and S. Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. C. H. Hoi. Deeptime: Deep time-index meta-learning for non-stationary time-series forecasting. *CoRR*, abs/2207.06046, 2022.

H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 22419–22430, 2021.

Y. Yin, M. Kirchmeyer, J.-Y. Franceschi, A. Rakotomamonjy, and P. Gallinari. Continuous pde dynamics forecasting with implicit neural representations. In *International Conference on Learning Representations, ICLR*, 2023.

A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting? *CoRR*, abs/2205.13504, 2022.

H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, pages 11106–11115, 2021.

T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 27268–27286. PMLR, 2022.

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019.

A Reproductiblity statement

Our work is entirely reproducible, and all the references to the information in order to reproduce it are in this section.

Code. The code for all our experiments is available at this link.

Data. A subset of the processed data is available with the code at this link. The dataset description, processing and normalization are presented in Appendix C.

Model. The model and the training details are presented in Section 3 and the hyperparameter selection is available in Appendix B.1.

GPU. We used NVIDIA TITAN RTX 24Go single GPU to conduct all the experiments for our method, which is coded in PyTorch (Python 3.9.2).

B Architecture details and ablation studies

B.1 Architecture details

For all imputation and forecasting experiments we choose the following hyperparameters :

- z dimension: 128
- Number of layers: 5
- Hidden layers dimension: 256
- $\gamma(t) \in \mathbb{R}^{2\times 64}$
- z code learning rate (α in Algorithm 1): 10^{-2}
- Hypernetwork and INR learning rate: 5×10^{-4}
- Number of steps in inner loop: $K = 3$
- Number of epochs: 4×10^4
- Batch size: 64

It is worth noting that the hyperparameters mentioned above remain consistent across all experiments conducted in the paper. We chose to maintain a fixed set of hyperparameters for our model, while other imputation and forecasting approaches commonly fine-tune hyperparameters based on a validation dataset. The obtained results exhibit high robustness across various settings, suggesting that the selected hyperparameters are already effective in achieving reliable outcomes.

B.2 Ablation studies

B.2.1 Fourier features vs SIREN on imputation task

Baseline The SIREN network differs from the Fourier features network because it does not explicitly incorporate frequencies as input. Instead, it is a multi-layer perceptron network that utilizes sine activation functions. An adjustable parameter, denoted ω_0, is multiplied with the input matrices of the preceding layers to capture a broader range of frequencies. For this comparison, we adopt the same hyperparameters described in Appendix B.1, selecting $\omega_0 = 30$ to align with Sitzmann et al. (2020). Furthermore, we set the learning rate of both the hypernetwork and the INR to 5×10^{-5} to enhance training stability. In Table 4, we compare the imputation results obtained by the Fourier features network and the SIREN network, specifically focusing on the first time window from the *Electricity*, *Traffic* and *Solar* datasets.

Table 4: MAE imputation errors on the first time window of each dataset. Best results are bold.

	τ	TimeFlow	TimeFlow w SIREN
Electricity	0.05	**0.323**	0.466
	0.10	**0.252**	0.350
	0.20	**0.224**	0.242
	0.30	**0.211**	0.222
	0.50	**0.194**	0.209
Solar	0.05	**0.105**	0.114
	0.10	**0.083**	0.094
	0.20	**0.065**	0.079
	0.30	**0.061**	0.072
	0.50	**0.056**	0.066
Traffic	0.05	**0.292**	0.333
	0.10	**0.220**	0.252
	0.20	**0.168**	0.191
	0.30	**0.152**	0.163
	0.50	**0.141**	0.154

Results According to the results presented in Table 4, the Fourier features network outperforms the SIREN network in the imputation task on these datasets. Notably, the performance gap between the two network architectures are more pronounced at low sampling rates. This disparity can be attributed to the SIREN network's difficulty in accurately capturing high frequencies when the time series is sparsely observed. We hypothesize that the MLP with ReLU activations correctly learns the different frequencies of time series with multi-temporal patterns by switching on or off the Fourier embedding frequencies.

B.2.2 Influence of the latent code dimension

The dimension of the latent code z is a crucial parameter in our architecture. If it is too small, it underfits the time series. Consequently, this adversely affects the performance of both the imputation and forecasting tasks. On the other hand, if the dimension of z is too large, it can lead to overfitting, hindering the model's ability to generalize to new data points.

Baselines To investigate the impact of z dimensionality on the performance of TimeFlow, we conducted experiments on the three considered datasets, specifically focusing on the forecasting task. We varied the sizes of z within $\{32, 64, 128, 256\}$. The other hyperparameters are set as presented in Appendix B.1. The obtained results for each z dimension are summarized in Table 5.

Table 5: MAE error for different z dimension.

	H	32	64	128	256
Electricity	96	0.232 ± 0.016	0.222 ± 0.017	0.222 ± 0.018	**0.215 ± 0.019**
	192	0.245 ± 0.020	0.239 ± 0.018	**0.230 ± 0.026**	0.233 ± 0.017
	336	0.254 ± 0.029	0.244 ± 0.028	0.262 ± 0.031	**0.243 ± 0.032**
	720	0.295 ± 0.027	0.284 ± 0.028	0.303 ± 0.041	**0.283 ± 0.029**
SolarH	96	0.182 ± 0.009	0.181 ± 0.012	**0.179 ± 0.003**	0.225 ± 0.047
	192	0.195 ± 0.014	0.195 ± 0.016	**0.193 ± 0.015**	0.197 ± 0.029
	336	**0.181 ± 0.011**	0.182 ± 0.011	0.189 ± 0.013	0.183 ± 0.012
	720	0.201 ± 0.027	**0.199 ± 0.025**	0.209 ± 0.029	0.200 ± 0.030
Traffic	96	0.223 ± 0.024	0.215 ± 0.028	0.215 ± 0.037	**0.210 ± 0.033**
	192	0.214 ± 0.018	0.217 ± 0.025	0.206 ± 0.023	**0.203 ± 0.024**
	336	0.238 ± 0.029	0.231 ± 0.029	**0.226 ± 0.030**	0.229 ± 0.029
	720	0.272 ± 0.040	0.269 ± 0.035	**0.259 ± 0.038**	0.262 ± 0.040

Results The results presented in Table 5 suggest that a z dimension of 128 is a reasonable compromise but only optimal for some settings. Moreover, even though the choice of z dimension seems important, it doesn't critically impact the MAE error for the forecasting task.

B.2.3 Influence of the number of gradient steps

As can be seen in Table 6, using three gradient steps at inference yield an inference time of less than 0.2 seconds. The latter can still be reduced by doing only one step at the cost of an increase in the forecasting error. As observed in Table 6, increasing the number of gradient steps above 3 steps during inference does not improve forecasting performance.

Table 6: Inference time (in seconds) and MAE error on the forecasting task on the *Electricity* dataset for a horizon of length 720, a look-back window of length 512, and a varying number of adaptation gradient steps. The statistics are computed over 10 runs using an NVIDIA TITAN RTX GPU.

Gradient descent steps	1	3	10	50	500	5000
Inference time (s)	0.109 ± 0.003	0.176 ± 0.009	0.427 ± 0.031	3.547 ± 0.135	17.722 ± 0.536	189.487 ± 8.060
MAE	0.351 ± 0.038	0.303 ± 0.041	0.300 ± 0.040	0.299 ± 0.039	0.302 ± 0.038	0.308 ± 0.037

Table 7: MAE error on the forecasting task using 1 inner-step during training and a varying number of adaptation gradient steps at inference. Best results are in bold and / symbol means that the MAE score is very high (≥ 1).

	H	1	3	10	50
Electricity	96	**0.244 ± 0.017**	0.246 ± 0.017	0.261 ± 0.016	/
	192	**0.253 ± 0.024**	**0.253 ± 0.022**	0.261 ± 0.020	0.265 ± 0.019
	336	**0.267 ± 0.032**	0.268 ± 0.030	0.277 ± 0.028	0.281 ± 0.027
	720	0.302 ± 0.030	0.306 ± 0.029	0.310 ± 0.028	**0.301 ± 0.029**
SolarH	96	**0.192 ± 0.023**	0.623 ± 0.397	/	/
	192	**0.175 ± 0.006**	0.252 ± 0.068	/	/
	336	**0.192 ± 0.016**	0.471 ± 0.029	/	/
	720	**0.216 ± 0.034**	0.465 ± 0.063	/	0.550 ± 0.187
Traffic	96	**0.215 ± 0.029**	0.329 ± 0.039	/	/
	192	**0.208 ± 0.019**	0.310 ± 0.033	0.312 ± 0.032	/
	336	**0.237 ± 0.028**	0.307 ± 0.038	/	/
	720	**0.263 ± 0.038**	0.320 ± 0.040	/	/

Table 8: MAE error on the forecasting task using 3 inner-steps during training and a varying number of adaptation gradient steps at inference. Best results are in bold.

	H	1	3	10	50
Electricity	96	0.259 ± 0.020	**0.222 ± 0.018**	**0.222 ± 0.017**	0.228 ± 0.019
	192	0.269 ± 0.020	**0.230 ± 0.026**	0.232 ± 0.020	0.233 ± 0.026
	336	0.273 ± 0.033	**0.262 ± 0.031**	0.264 ± 0.032	0.268 ± 0.032
	720	0.351 ± 0.038	0.303 ± 0.041	0.300 ± 0.040	**0.299 ± 0.039**
SolarH	96	0.487 ± 0.196	**0.179 ± 0.003**	0.181 ± 0.003	0.186 ± 0.003
	192	0.411 ± 0.088	**0.193 ± 0.015**	0.195 ± 0.014	0.199 ± 0.013
	336	0.435 ± 0.153	**0.189 ± 0.013**	0.203 ± 0.006	0.223 ± 0.012
	720	0.394 ± 0.173	0.209 ± 0.029	**0.203 ± 0.006**	0.209 ± 0.027
Traffic	96	0.320 ± 0.038	**0.215 ± 0.037**	0.219 ± 0.043	0.226 ± 0.046
	192	0.299 ± 0.023	**0.206 ± 0.023**	0.209 ± 0.026	0.214 ± 0.027
	336	0.345 ± 0.038	**0.226 ± 0.030**	0.228 ± 0.031	0.233 ± 0.032
	720	0.321 ± 0.034	**0.259 ± 0.038**	0.260 ± 0.038	0.266 ± 0.039

Results We conduct more extensive experiments in Table 7, Table 8, Table 9 to quantify the MAE score variation according to different number of gradient steps during training and inference. The tables show that using the same number of steps in training and inference leads to better results. This is expected since using different gradient steps for training and inference makes the inference model slightly different from the training model. In addition, using 3 gradient steps instead of 1 clearly improves the performances, but using 10 instead of 3 does not. Indeed, it usually leads to overall better results for longer horizon, but the gain is not clear for smaller horizons. Hence using 3 gradient steps is a suitable choice.

B.2.4 TimeFlow variants with other meta-learning techniques

Baselines Before converging to the current architecture and optimization of TimeFlow, we explored different options to condition the INR with the observations. The first one was inspired by the neural process

Table 9: MAE error on the forecasting task using 10 inner-steps during training and a varying number of adaptation gradient steps at inference. Best results are in bold.

	H	1	3	10	50
Electricity	96	0.381 ± 0.030	0.249 ± 0.024	**0.236 ± 0.024**	0.238 ± 0.024
	192	0.448 ± 0.045	0.273 ± 0.019	**0.244 ± 0.014**	**0.244 ± 0.013**
	336	0.514 ± 0.053	0.283 ± 0.033	**0.241 ± 0.025**	0.242 ± 0.024
	720	0.647 ± 0.068	0.400 ± 0.051	**0.286 ± 0.023**	0.287 ± 0.021
SolarH	96	0.605 ± 0.029	0.380 ± 0.018	**0.188 ± 0.012**	0.199 ± 0.015
	192	0.382 ± 0.072	0.250 ± 0.012	**0.202 ± 0.034**	0.204 ± 0.035
	336	0.745 ± 0.105	0.431 ± 0.221	**0.201 ± 0.033**	0.208 ± 0.032
	720	0.745 ± 0.082	0.477 ± 0.039	**0.205 ± 0.030**	**0.205 ± 0.029**
Traffic	96	0.450 ± 0.023	0.273 ± 0.026	**0.225 ± 0.028**	0.230 ± 0.034
	192	0.506 ± 0.028	0.318 ± 0.021	**0.233 ± 0.022**	0.236 ± 0.026
	336	0.500 ± 0.042	0.320 ± 0.021	**0.247 ± 0.028**	0.249 ± 0.031
	720	0.511 ± 0.035	0.323 ± 0.022	**0.266 ± 0.027**	0.272 ± 0.024

architecture, which uses a set encoder to transform a set of observations $(t_i, x_{t_i})_{i \in \mathcal{I}}$ into a latent code z by applying a pooling layer after a feed forward network. We observed that this encoder in combination with the modulated fourier features network was able to achieve relatively good results on the forecasting task but suffered of underfitting on more complex datasets such as *Electricity*.

This led us to consider auto-decoding methods instead, i.e. encoder-less, architectures for conditioning the weights of the coordinate-based network. We trained TimeFlow with the REPTILE algorithm (Nichol et al., 2018), which is a first-order meta-learning technique that adapts the code in a few steps of gradient descent. In contrast with a second-order method, we observed that REPTILE was less costly to train but struggled to escape sub optimal minima, which led to unstable training and underfitting.

From an implementation point of view, the only difference between second order and first order, is that in the latter the code is detached from the computation graph before taking the outer-loop parameter update. When the code is not detached, it remains a function of the common parameters $z = z(\theta, w)$, which means that the computation graph for the outer-loop also includes the inner-loop updates to the codes. Therefore the outer-loop gradient update involves a gradient through a gradient and requires an additional backward pass through the INR to compute the Hessian. Please refer to Finn et al. (2017) for more technical details.

Table 10: Comparison of second-order and first-order (REPTILE) meta learning for TimeFlow on the imputation task. Mean MAE results on the missing grid over five different time windows. τ stands for the subsampling rate. Bold results are best.

	τ	TimeFlow	TimeFlow w REPTILE
Electricity	0.05	**0.324 ± 0.013**	0.363 ± 0.062
	0.10	**0.250 ± 0.010**	0.343 ± 0.036
	0.20	**0.225 ± 0.008**	0.312 ± 0.043
	0.30	**0.212 ± 0.007**	0.308 ± 0.035
	0.50	**0.194 ± 0.007**	0.305 ± 0.046
Solar	0.05	**0.095 ± 0.015**	0.125 ± 0.025
	0.10	**0.083 ± 0.015**	0.123 ± 0.032
	0.20	**0.072 ± 0.015**	0.108 ± 0.021
	0.30	**0.061 ± 0.012**	0.105 ± 0.027
	0.50	**0.054 ± 0.013**	0.102 ± 0.021
Traffic	0.05	**0.283 ± 0.016**	0.304 ± 0.026
	0.10	**0.211 ± 0.012**	0.264 ± 0.009
	0.20	**0.168 ± 0.006**	0.242 ± 0.019
	0.30	**0.151 ± 0.007**	0.218 ± 0.020
	0.50	**0.139 ± 0.007**	0.216 ± 0.017

Results In Table 10, we show the performance of first-order TimeFlow on the imputation task. In low sampling regimes the difference with TimeFlow is less perceptive, but its performance plateaus when the number of points increases. This is not surprising. Indeed, as though the task is actually simpler when τ increases, the optimization is made more difficult with the increased number of observations. We provide the performance of TimeFlow with a set encoder on the Forecasting task in Table 11. We observed that this version failed to generalize well for complex datasets.

Table 11: Comparison of optimization-based and set-encoder-based meta learning for TimeFlow on the forecasting task. Mean MAE forecast results over different time windows. H stands for the horizon. Bold results are best.

	H	TimeFlow	TimeFlow w set encoder
Electricity	96	**0.228 ± 0.026**	0.362 ± 0.032
	192	**0.238 ± 0.020**	0.360 ± 0.028
	336	**0.270 ± 0.031**	0.382 ± 0.038
	720	**0.316 ± 0.055**	0.431 ± 0.059
SolarH	96	**0.190 ± 0.013**	0.251 ± 0.071
	192	**0.202 ± 0.020**	0.239 ± 0.058
	336	**0.209 ± 0.017**	0.235 ± 0.040
	720	**0.218 ± 0.048**	0.231 ± 0.032
Traffic	96	**0.217 ± 0.036**	0.276 ± 0.031
	192	**0.212 ± 0.028**	0.281 ± 0.034
	336	**0.238 ± 0.034**	0.297 ± 0.042
	720	**0.279 ± 0.050**	0.333 ± 0.048

B.2.5 Influence of the modulation

In TimeFlow, we apply shift modulations to the parameters of the INR, i.e. for each layer l we only modify the biases of the network with an extra bias term $\phi_l^{(j)}$. We generate these bias terms with a linear hypernetwork that maps the code $z^{(j)}$ to the modulations. The output of the l-th layer of the modulated INR is thus given by $\phi_{l+1} = \text{ReLU}(\theta_l \phi_{l-1} + b_l + \psi_l^{(j)})$, where $\psi_l^{(j)} = W_l z^{(j)}$ and $(W_l)_{l=1}^L$ are parameters of the hypernetwork. However, another common modulation is the combination of the scale and shift modulation, which leads to the output of the l-th layer of the modulated INR being given by $\phi_{l+1} = \text{ReLU}((S_l z^{(j)}) \circ (\theta_l \phi_{l-1} + b_l) + \psi_l^{(j)})$, where $\psi_l^{(j)} = W_l z^{(j)}$, and $(W_l)_{l=1}^L$ and $(S_l)_{l=1}^L$ are parameters of the hypernetwork and $\circ$ is the Hadamard product.

In Table 12, we conduct additional experiments on the *Electricity* dataset in the forecasting setting with different time horizons. In these experiments, we compare two scenarios: one where the INR is modulated only by a shift factor and the other where the INR is modulated by both a shift and a scale factor. We kept the architecture and hyperparameters consistent with those described in Appendix B.1. The experiments shown in Table 12 indicate that the INR is longer to train with shift and scale modulations due to the increased number of parameters involved. Furthermore, we observe that the shift and scale modulated INR performed similarly or even worse than the INR with only shift modulation. These two drawbacks, namely an increased computational time and similar or worse performances, motivate modulating the INR only by a shift factor.

Table 12: Ablation on modulations for the forecasting task on *Electricity* dataset for different horizons. Models are trained on a given time window and tested on four new time windows. Models are trained on a single NVIDIA TITAN RTX GPU.

	96		192		336		720	
	MAE	Training time	MAE	Training time	MAE	Training time	MAE	Training time
Shift	0.233 ± 0.014	2h30	0.245 ± 0.016	2h31	0.264 ± 0.020	2h33	0.303 ± 0.041	2h46
Shift and scale	0.257 ± 0.019	3h29	0.263 ± 0.014	3h32	0.268 ± 0.025	3h45	0.308 ± 0.037	4h14

B.2.6 Discussion on other hyperparameters

While the dimension of z is indeed a crucial hyperparameter, it is important to note that other hyperparameters also play a significant role in the performance of the INR. For example, the number of layers in the FFN directly affects the ability of the model to fit the time series. In our experiments, we have observed that using five or more layers yields good performance, and including additional layers can lead to slight improvements in the generalization settings.

Similarly, the number of frequencies used in the frequency embedding is another important hyperparameter. Using too few frequencies can limit the network's ability to capture patterns, while using too many frequencies can hinder its ability to generalize accurately.

The choice of learning rate is critical for achieving stable convergence during training. Therefore, in practice, we use a low learning rate combined with a cosine annealing scheduler to ensure stable and effective training.

C Datasets, scores and normalization

For the complete datasets, *Electricity* dataset is available here, *Traffic* dataset here and *Solar* data set here. Table 13 provides a concise overview of the main information about the datasets used for forecasting and imputation tasks.

Table 13: Summary of datasets information

Dataset name	Number of samples	Number of time steps	Sampling frequency	Location	Years
Electricity	321	26 304	hourly	Portugal	2012 – 2014
Traffic	862	17 544	hourly	San Francisco bay	2015 – 2016
Solar	137	52 560	10 minutes	Alabama	2006
SolarH	137	8 760	hourly	Alabama	2006

How TimeFlow relative improvement score is computed? In many paper tables, the TimeFlow improvement score appears in the last row of the table. Its purpose is to quantify the average marginal gain of TimeFlow over the method under consideration. It is computed as follows:

$$\text{TimeFlow improvement} = \frac{1}{L}\sum_{l=1}^{L}\frac{s_l(\text{baseline}) - s_l(\text{TimeFlow})}{s_l(\text{baseline})}$$

- s stands for the Mean Absolute Error score of the considered method against the ground truth at line l (for instance in Table 1, s_1(TimeFlow)= 0.324, s_2(TimeFlow)= 0.250 etc).
- L stands for the number of line in the table (for instance 15 in Table 1).

z-normalization To preprocess each dataset, we apply the widely used z-normalization technique per-sample j on the entire series: $x_{norm}^{(j)} = \frac{x^{(j)} - \text{mean}(x^{(j)})}{\text{std}(x^{(j)})}$.

D Imputation experiments

D.1 Baselines details

D.1.1 Baselines training and hyperparameters

The baselines underwent meticulous training and extensive testing, involving thorough exploration of hyperparameters. We used SAITS repository (code) for BRITS and SAITS. The adopted setting results from a hyperparameters search, which yields marginally superior results for both methods compared to the default settings. The marginal difference in scores underscores the robustness of BRITS and SAITS. In addition, mTAN and TIDER did not perform optimally with the recommended configurations, requiring an extensive search. Details of the parameters explored are provided in Table 14 and Table 15. While the hyperparameter search led to performance improvements, overall results remained sub-optimal. For CSDI, the recommended settings proved inadequate for the considered datasets, prompting a comprehensive search. Among various parameters, the number of diffusion steps emerged as crucial, significantly enhancing performance, particularly at higher draw ratios. However, superior performance was attained with more diffusion steps, albeit at increased computational cost. The chosen parameters are detailed in Table 16. In addition, the original DeepTime implementation did not perform well on imputation. Hence, we adapted the DeepTime

training procedure (see Appendix D.3). Lastly, the vanilla Neural Process baseline underperformed, so we customized its architecture to conduct a fair comparison with TimeFlow. We used the INR and hypernetwork from TimeFlow to align the Neural Process with our temporal frequency bias and shift modulation technique.

Table 14: mTAN hyperparameter search.

Dimension size	γ linear scheduler	lr	NumRefPoints	k-iwae	Target ratio
50	1	1×10^{-5}	32	5	0.2
100	0.95	0.0001	64	10	0.8
-	0.5	0.001	128	-	-
-	0.1	0.005	-	-	-

Table 15: TIDER hyperparameter search.

Dimension size	λ_{ar}	λ_{trend}	lr	Season number
50	0.1	0.01	0.0001	2
100	0.2	0.1	0.001	10
-	-	-	0.005	15
-	-	-	-	20

Table 16: CSDI chosen hyperparameters.

Epochs	lr	Layers	Channels	Nheads	Diffusion embedding dimension	NSteps	Schedule	Time embedding	Feature embedding
5000	0.001	4	64	8	128	100	Quad	128	16

D.1.2 Models complexity

We can see in Table 17 that our method has 10 times less parameters than BRITS and 20 times less than SAITS. It is mainly due to their modelisation of interaction between samples. SAITS, which is based on transformers has the highest number of parameters when mTAN has the lowest number of parameters.

Table 17: Number of parameters for each DL methods on the imputation task on the *Electricity* dataset.

	TimeFlow	DeepTime	NeuralProcess	mTAN	SAITS	BRITS	TIDER
Number of parameters	602k	1315k	248k	113k	11 137k	6 220k	1 034k

D.2 Imputation for previously unseen time series

Setting In this section we analyze in details the imputations results for previously unseen time series described in Section 4.1. Specifically, TimeFlow is trained on a given set of time series within a defined time window and then used for inference on new time series. We train TimeFlow on 50 % of the samples and consider the remaining 50 % as the new time series.

We compare in Table 18 observed grid fit scores and missing grid inference scores for time series known at training and time series unknown at training.

Results The results presented in Table 18 indicate that the inference MAE for missing grids shows consistency between known and new samples, regardless of the data or sampling rate. However, it is worth noting that there is a slight drop in performance compared to the results in table Table 1. This decrease is because in Table 18, the shared architecture is trained on only half the samples, affecting its overall performance.

Table 18: TimeFlow MAE imputation errors results for imputation previsouly unseen time series.

		Known time series		New time series	
	τ	Fit	Inference	Fit	Inference
Electricity	0.05	0.060 ± 0.010	0.402 ± 0.021	0.142 ± 0.083	0.413 ± 0.026
	0.10	0.046 ± 0.006	0.302 ± 0.010	0.144 ± 0.098	0.309 ± 0.016
	0.20	0.067 ± 0.015	0.285 ± 0.014	0.154 ± 0.089	0.291 ± 0.022
	0.30	0.093 ± 0.022	0.266 ± 0.010	0.163 ± 0.073	0.271 ± 0.017
	0.50	0.108 ± 0.012	0.236 ± 0.010	0.167 ± 0.061	0.245 ± 0.017
Solar	0.05	0.014 ± 0.002	0.104 ± 0.015	0.050 ± 0.037	0.109 ± 0.016
	0.10	0.017 ± 0.002	0.092 ± 0.015	0.052 ± 0.036	0.099 ± 0.017
	0.20	0.028 ± 0.008	0.078 ± 0.014	0.058 ± 0.031	0.089 ± 0.017
	0.30	0.038 ± 0.009	0.072 ± 0.013	0.063 ± 0.028	0.084 ± 0.018
	0.50	0.045 ± 0.011	0.066 ± 0.013	0.067 ± 0.025	0.080 ± 0.019
Traffic	0.05	0.044 ± 0.003	0.291 ± 0.013	094 ± 0.051	0.291 ± 0.012
	0.10	0.033 ± 0.001	0.209 ± 0.010	0.093 ± 0.060	0.216 ± 0.012
	0.20	0.037 ± 0.006	0.175 ± 0.008	0.095 ± 0.058	0.186 ± 0.013
	0.30	0.048 ± 0.005	0.164 ± 0.006	0.098 ± 0.051	0.175 ± 0.013
	0.50	0.068 ± 0.004	0.159 ± 0.007	0.110 ± 0.042	0.169 ± 0.012

D.3 Details on DeepTime adaptation for imputation

As DeepTime was proposed to address the forecasting task with a deeptime-index model, the authors did not tackle the task of imputation and left it out for future work. Given the success of this method and the motivation of our work, we wanted to explore its capabilities to impute time series with several subsampling rates. Following our current framework, we first tried to train the model in a self-supervised way, i.e. trying to reconstruct observations $x^{(j)} \in \mathcal{T}^{(j)}$ after the INR has been conditioned with the Ridge Regressor on the same set of observations, but discovered failure cases for $\tau \leq 0.20$. To be faithful to the original supervised training of DeepTime, we therefore randomly mask out 50% of the observations that we use as context for the Ridge Regressor and try to infer the other 50% (the targets) to train the INR.

We provide a qualitative comparison of the model's performance with these two different training procedures in Figure 7. We can notice that the model that results from the self-supervised training perfectly fits the observations but completely misses the important patterns of the series. On the other hand, when DeepTime is trained to infer target values based on observations, it is able to capture the general trends. We think that in the small subsampling regime ($\tau \leq 0.20$), the Ridge Regressor easily fits very well all the observations which hinders the training of the INR's basis.

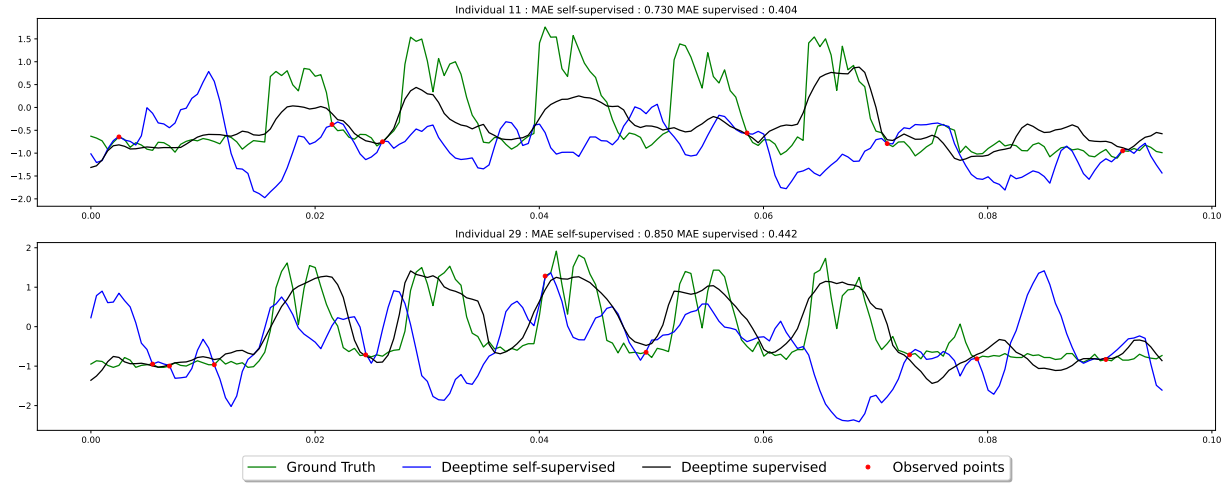


Figure 7: *Electricity dataset.* Self supervised DeepTime imputation (blue line) and supervised DeepTime imputation (black line) with 5% of known point (red points) on the eight first days of samples 11 (top) and 29 (bottom).

D.4 Imputation against non deep learning methods

Setting In addition to the deep learning methods presented in Table 1, we evalute TimeFlow against two classic machine learning baselines, K-Nearest Neighbours (KNN) and linear interpolation, which are valuable for getting an idea of the complexity of the problem.

Table 19: Mean MAE imputation results on the missing grid only over five different time window. τ stands for the subsampling rate. Bold results are best, underline results are second best.

	τ	TimeFlow	Linear interpolation	KNN (k=3)
Electricity	0.05	**0.324 ± 0.013**	0.828 ± 0.045	0.531 ± 0.033
	0.10	**0.250 ± 0.010**	0.716 ± 0.039	0.416 ± 0.020
	0.20	**0.225 ± 0.008**	0.518 ± 0.029	0.363 ± 0.019
	0.30	**0.212 ± 0.007**	0.396 ± 0.022	0.342 ± 0.017
	0.50	**0.194 ± 0.007**	0.275 ± 0.015	0.323 ± 0.016
Solar	0.05	**0.095 ± 0.015**	0.339 ± 0.031	0.151 ± 0.017
	0.10	**0.083 ± 0.015**	0.170 ± 0.014	0.128 ± 0.017
	0.20	**0.072 ± 0.015**	0.088 ± 0.010	0.110 ± 0.016
	0.30	**0.061 ± 0.012**	0.063 ± 0.009	0.103 ± 0.017
	0.50	0.054 ± 0.013	**0.044 ± 0.008**	0.096 ± 0.016
Traffic	0.05	**0.283 ± 0.016**	0.813 ± 0.027	0.387 ± 0.014
	0.10	**0.211 ± 0.012**	0.701 ± 0.026	0.293 ± 0.012
	0.20	**0.168 ± 0.006**	0.508 ± 0.022	0.249 ± 0.010
	0.30	**0.151 ± 0.007**	0.387 ± 0.018	0.228 ± 0.009
	0.50	**0.139 ± 0.007**	0.263 ± 0.013	0.204 ± 0.009
TimeFlow improvement		/	49.06 %	35.95 %

Results KNN imputation uses information from other individuals and gives satisfactory results at all sampling rates. On the other hand, the purely univariate approach of linear interpolation struggles at low sampling rates but performs well at high sampling rates. TimeFlow significantly outperforms both baselines by a large margin.

E Forecasting experiments

E.1 Distinction between adjacent time windows and new time windows during inference

In Section 4.2, we presented the forecasting results for periods outside the training period. These periods can be classified into two types: adjacent to or disjoint from the training period. Figure 8 illustrates these distinct test periods for the *Electricity* dataset. The same principle applies to the *Traffic* and *SolarH* datasets, with one notable difference: the number of test periods is smaller in these datasets compared to *Electricity* dataset due to the fewer time steps available.

In Table 2, we presented the results indistinctly for the two types of test periods: adjacent to and disjoint from the training window. Here, we aim to differentiate the results for these two types of window and emphasize their significant impact on Informer and AutoFormer results. Specifically, Table 20 showcases the results for the test periods adjacent to the training window. In contrast, Table 21 displays the results for the test periods disjointed from the training window

Results TimeFlow, PatchTST, DLinear and DeepTime maintain consistent forecasting results whether tested on the period adjacent to the training period or on a disjoint period. However, AutoFormer and Informer show a significant drop in performance when tested on new disjoint periods.

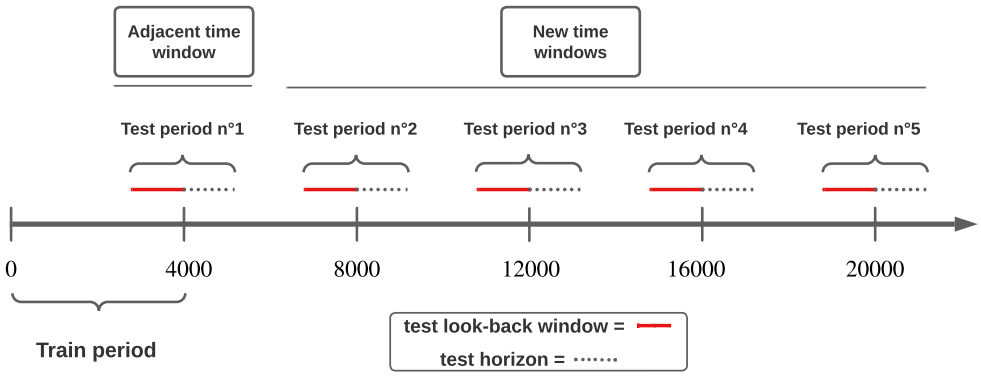


Figure 8: Distinction between adjacent time windows and new time windows during inference for the *Electricity* dataset

Table 20: Mean MAE forecast results for adjacent time windows. H stands for the horizon. Bold results are best, underline results are second best.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.218 ± 0.017	0.240 ± 0.027	0.392 ± 0.045	**0.214 ± 0.020**	0.236 ± 0.035	0.310 ± 0.031	0.293 ± 0.0184
	192	0.238 ± 0.012	0.251 ± 0.023	0.401 ± 0.046	**0.225 ± 0.017**	0.248 ± 0.032	0.322 ± 0.046	0.336 ± 0.032
	336	0.265 ± 0.036	0.290 ± 0.034	0.434 ± 0.075	**0.242 ± 0.024**	0.284 ± 0.043	0.330 ± 0.019	0.405 ± 0.044
	720	0.318 ± 0.073	0.356 ± 0.060	0.605 ± 0.149	**0.291 ± 0.040**	0.370 ± 0.086	0.456 ± 0.052	0.489 ± 0.072
SolarH	96	**0.172 ± 0.017**	0.197 ± 0.002	0.221 ± 0.048	0.232 ± 0.008	0.204 ± 0.002	0.261 ± 0.053	0.273 ± 0.023
	192	**0.198 ± 0.010**	0.202 ± 0.014	0.244 ± 0.048	0.231 ± 0.027	0.211 ± 0.012	0.312 ± 0.085	0.256 ± 0.026
	336	0.207 ± 0.019	**0.200 ± 0.012**	0.241 ± 0.005	0.254 ± 0.048	0.212 ± 0.019	0.341 ± 0.107	0.287 ± 0.006
	720	**0.215 ± 0.016**	0.240 ± 0.011	0.403 ± 0.147	0.271 ± 0.036	0.246 ± 0.015	0.368 ± 0.006	0.341 ± 0.049
Traffic	96	0.216 ± 0.033	0.229 ± 0.032	0.283 ± 0.028	**0.201 ± 0.031**	0.225 ± 0.034	0.299 ± 0.080	0.324 ± 0.113
	192	0.208 ± 0.021	0.220 ± 0.020	0.292 ± 0.023	**0.195 ± 0.024**	0.215 ± 0.022	0.320 ± 0.036	0.321 ± 0.052
	336	0.237 ± 0.040	0.247 ± 0.033	0.305 ± 0.039	**0.220 ± 0.036**	0.244 ± 0.035	0.450 ± 0.127	0.394 ± 0.066
	720	**0.266 ± 0.048**	0.290 ± 0.045	0.339 ± 0.037	0.268 ± 0.050	0.290 ± 0.047	0.630 ± 0.043	0.441 ± 0.055
TimeFlow improvement		/	6.56 %	30.79 %	2.64 %	7.30 %	35.43 %	33.07 %

Table 21: Mean MAE forecast results for new time windows. H stands for the horizon. Bold results are best, underline results are second best.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.230 ± 0.012	0.245 ± 0.026	0.392 ± 0.045	**0.222 ± 0.023**	0.240 ± 0.025	0.606 ± 0.281	0.605 ± 0.227
	192	0.246 ± 0.025	0.252 ± 0.018	0.401 ± 0.046	**0.231 ± 0.020**	0.257 ± 0.027	0.545 ± 0.186	0.776 ± 0.257
	336	0.271 ± 0.029	0.285 ± 0.034	0.434 ± 0.076	**0.253 ± 0.027**	0.298 ± 0.051	0.571 ± 0.181	0.823 ± 0.241
	720	0.316 ± 0.051	0.359 ± 0.048	0.607 ± 0.15	**0.299 ± 0.038**	0.373 ± 0.075	0.674 ± 0.245	0.811 ± 0.257
SolarH	96	0.208 ± 0.005	**0.206 ± 0.026**	0.221 ± 0.048	0.293 ± 0.089	0.212 ± 0.019	0.228 ± 0.027	0.234 ± 0.011
	192	**0.206 ± 0.012**	0.207 ± 0.037	0.244 ± 0.048	0.274 ± 0.060	0.223 ± 0.029	0.356 ± 0.122	0.280 ± 0.033
	336	0.211 ± 0.005	**0.199 ± 0.035**	0.240 ± 0.006	0.264 ± 0.088	0.223 ± 0.032	0.327 ± 0.029	0.366 ± 0.039
	720	**0.222 ± 0.020**	0.217 ± 0.028	0.403 ± 0.147	0.262 ± 0.083	0.251 ± 0.047	0.335 ± 0.075	0.333 ± 0.012
Traffic	96	0.218 ± 0.042	0.229 ± 0.032	0.283, 0.0275	**0.204 ± 0.039**	0.229 ± 0.032	0.326 ± 0.049	0.388 ± 0.055
	192	0.213 ± 0.028	0.220 ± 0.023	0.292, 0.0236	**0.198 ± 0.031**	0.223 ± 0.023	0.575 ± 0.254	0.381 ± 0.049
	336	0.239 ± 0.035	0.244 ± 0.040	0.305, 0.0392	**0.223 ± 0.040**	0.252 ± 0.042	0.598 ± 0.286	0.448 ± 0.055
	720	0.280 ± 0.047	0.290 ± 0.055	0.339, 0.0375	**0.270 ± 0.059**	0.304 ± 0.061	0.641 ± 0.072	0.468 ± 0.064
TimeFlow improvement		/	2.50 %	27.75 %	3.41 %	6.80 %	46.26 %	45.53 %

E.2 Plots comparison: TimeFlow vs PatchTST

Table 2 demonstrates the similar forecasting performance of TimeFlow and PatchTST across all horizons. To visually represent their predictions, the figures below showcase the forecasted outcomes of these methods for two samples (24 and 38) and two horizons (96 and 192) on the *Electricity*, *SolarH*, and *Traffic* datasets.

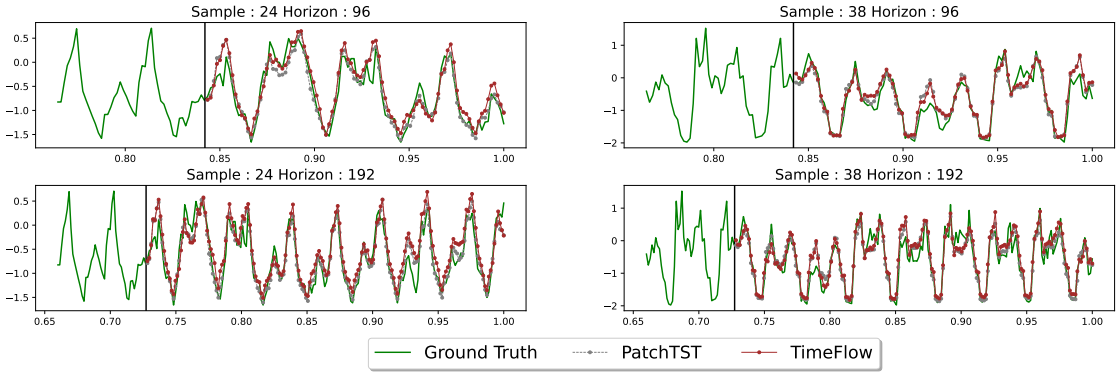


Figure 9: Qualitative comparisons of TimeFlow vs PatchTST on the *Electricity* dataset for new time windows

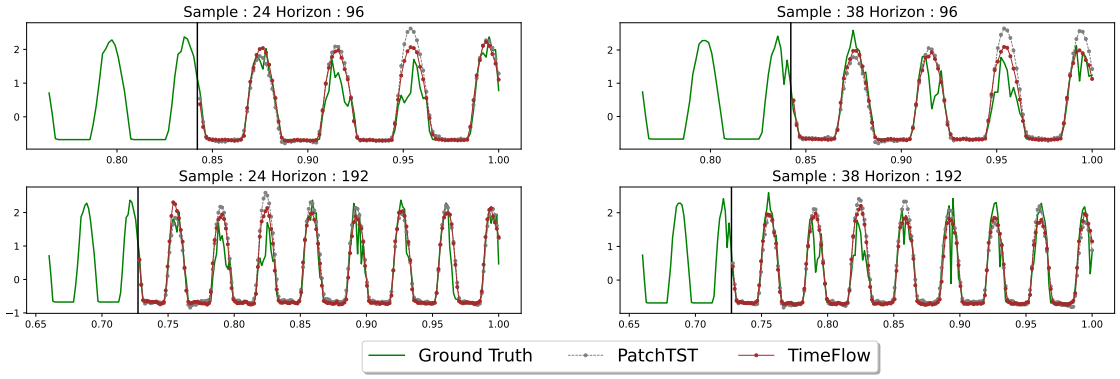


Figure 10: Qualitative comparisons of TimeFlow vs PatchTST on the *SolarH* dataset for new time windows

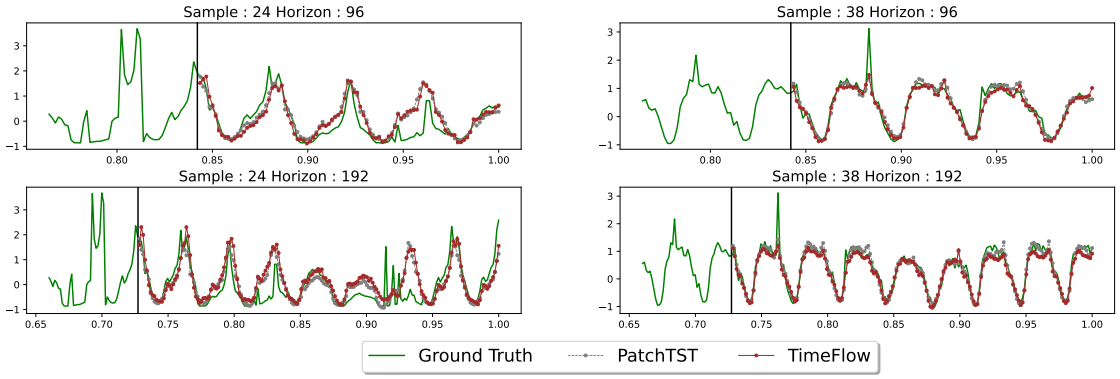


Figure 11: Qualitative comparisons of TimeFlow vs PatchTST on the *Traffic* dataset for new time windows

Results The visual analysis of the figures above reveals that the predictions of TimeFlow and PatchTST are remarkably similar. For instance, when examining sample 24 and horizon 192 of the *Traffic* dataset, both

forecasters exhibit similar error patterns. The only noticeable distinction emerges in the *SolarH* dataset, where PatchTST tends to overestimate certain peaks.

E.3 Baseline details

E.3.1 Baselines training and hyperparameters

We provide a detailed breakdown of the hyperparameters and our training approach for the forecasting baselines. We took an in-depth approach, testing each method under a range of configurations to ensure they were well-suited to the unique characteristics of the datasets and tasks at hand. For DLinear and transformer baselines, including PatchTST, AutoFormer, and Informer, we utilized the implementations detailed in the PatchTST baselines (code) and adhered to the best practices recommended for our particular tasks. Notably, our implementation of PatchTST was combined with ReVIN, enhancing the robustness of the results. Regarding DeepTime, we followed the recommended hyperparameters, opting for a structure with 5 layers, each 256 units wide, and using 4096 Fourier features spanning a diverse set of scales. As for the Neural Process, the standard model did not train as expected. So, we customized its architecture to conduct a fair comparison with TimeFlow. We used the INR and hypernetwork from TimeFlow to align the Neural Process with our temporal frequency bias and shift modulation technique. We also meticulously searched for the optimal hyperparameters, like the Kullback Leibler (KL) divergence weight and learning rate. Moreover, we extended the training duration to ensure thorough convergence.

E.3.2 Models complexity

In this section, we present the parameter counts and the inference time for the main forecasting baselines. Except for TimeFlow and DeepTime, the number of parameters varies with the number of samples, the look-back window, and the horizon. Thus, we report the number of parameters for two specific configurations, including a fixed dataset, a fixed look-back window, and a fixed horizon. In Table 22, we see that for PatchTST and DLinear, the larger the horizon, the more the number of parameters increases. In Table 23, it is shown that all methods' computational time increases with the horizon, which is expected. Moreover, TimeFlow is slower than the baselines that use forward computations only. Still, on the Electricity dataset, for example, the method can infer for 321 samples a horizon of 720 values with a look-back window of 512 timestamps in less than 0.2s, which does not look prohibitive for many real-world usages. This is mainly due to the small number of gradient steps at inference.

Table 22: The number of parameters for main baselines on the forecasting task on the *Electricity* dataset for horizons 96 and 720. The look-back window size is 512.

	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	Informer	Autoformer
96	602k	1 315k	480k	1 194k	98k	984k	1 005k
720	602k	1 315k	480k	6 306k	739k	984k	1 005k

Table 23: Inference time (in seconds) for the forecasting task on the *Electricity* dataset with horizons 96 and 720 and a look-back window of length 512. The statistics are computed over 10 runs using an NVIDIA TITAN RTX GPU.

	TimeFlow	Patch-TST	DLinear	DeepTime	AutoFormer	Informer
96	0.147 ± 0.007	0.016 ± 0.002	0.007 ± 0.003	0.006 ± 0.002	0.027 ± 0.001	0.0191 ± 0.002
720	0.176 ± 0.009	0.020 ± 0.001	0.009 ± 0.001	0.010 ± 0.002	0.034± 0.001	0.0251 ± 0.002

E.4 Sparsely observed look-back window: comparison with Patch-TST

Setting and baseline. Let's consider a setting where at inference time, the look-back window is sparsely observed. Models such as PatchTST must proceed in two steps: (i) completing the look-back window on a dense regular grid using imputation; (ii) apply the model on the completed window to predict the future. We compared TimeFlow with the following two-step processing baseline: linear interpolation handling the missing values within the partially observed look-back window, and PatchTST handling the forecasting task. We conducted experiments on the *Traffic* and *Electricity* datasets, focusing on the 96 and 192 horizons. In Table 24, we present the results at different sampling rates $\tau \in \{0.5, 0.2, 0.1\}$ within the look-back window.

Table 24: MAE results for forecasting on new samples and new period with missing values in the look-back window. Best results are in bold.

			TimeFlow		Linear interpo + PatchTST	
	H	τ	Imputation error	Forecast error	Imputation error	Forecast error
Electricity	96	1.	0.000 ± 0.000	0.228 ± 0.028	0.000 ± 0.000	**0.221 ± 0.023**
		0.5	**0.151 ± 0.003**	**0.239 ± 0.013**	0.257 ± 0.008	0.279 ± 0.026
		0.2	**0.208 ± 0.006**	**0.260 ± 0.015**	0.482 ± 0.019	0.451 ± 0.042
		0.1	**0.272 ± 0.006**	**0.295 ± 0.016**	0.663 ± 0.029	0.634 ± 0.053
	192	1.	0.000 ± 0.000	0.238 ± 0.020	0.000 ± 0.000	**0.229 ± 0.020**
		0.5	**0.149 ± 0.004**	**0.235 ± 0.011**	0.258 ± 0.006	0.280 ± 0.032
		0.2	**0.209 ± 0.006**	**0.257 ± 0.013**	0.481 ± 0.021	0.450 ± 0.054
		0.1	**0.274 ± 0.010**	**0.289 ± 0.016**	0.669 ± 0.030	0.650 ± 0.060
Traffic	96	1.	0.000 ± 0.000	0.217 ± 0.032	0.000 ± 0.000	**0.203 ± 0.037**
		0.5	**0.219 ± 0.017**	**0.224 ± 0.033**	0.276 ± 0.012	0.255 ± 0.041
		0.2	**0.278 ± 0.017**	**0.252 ± 0.029**	0.532 ± 0.017	0.483 ± 0.040
		0.1	**0.418 ± 0.019**	**0.382 ± 0.014**	0.738 ± 0.023	0.721 ± 0.073
	192	1.	0.000 ± 0.000	0.212 ± 0.028	0.000 ± 0.000	**0.197 ± 0.030**
		0.5	**0.176 ± 0.014**	**0.217 ± 0.017**	0.276 ± 0.011	0.245 ± 0.029
		0.2	**0.233 ± 0.017**	**0.236 ± 0.021**	0.532 ± 0.020	0.480 ± 0.050
		0.1	**0.304 ± 0.019**	**0.277 ± 0.021**	0.734 ± 0.022	0.787 ± 0.172

Results. Although PatchTST performs slightly better with a dense look-back window, its performance significantly deteriorates as the value of τ decreases. In contrast, the performance of TimeFlow is only minimally affected by the reduction in the sampling rate.

E.5 Influence of the look-back window for forecasting

In Figure 12, it is shown that both excessively short and overly long look-back windows can harm TimeFlow forecasting performance. More precisely, the performances increases with the look-back window size up to a certain size, where the performances then drop slowly.

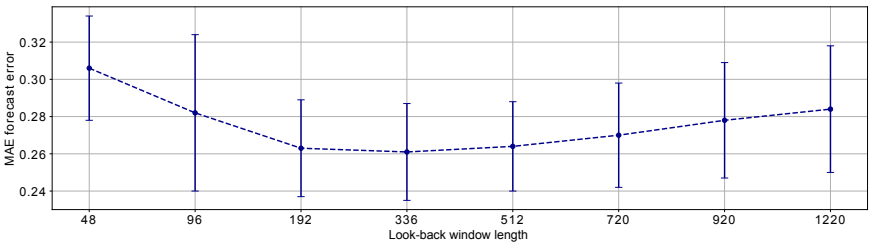


Figure 12: MAE forecast error per look-back windows length for the *Electricity* dataset (horizon window length is 336). The model is trained on a given time window and tested on four new time windows.

E.6 Influence of the horizon length for forecasting

In Figure 13, it is shown that the performances decrease with the length of the horizon. This is to be expected, since the longer the horizon, the harder the task.

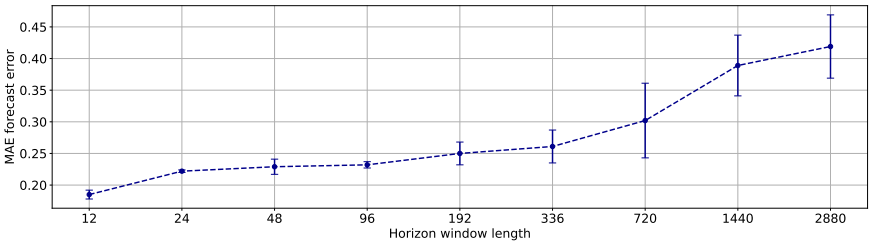


Figure 13: MAE forecast error per horizons length for the *Electricity* dataset (look-back window length is 512). The model is trained on a given time window and tested on four new time windows.

F Discussion on using frequency embedding as input to regression models

F.1 Related work

In forecasting univariate time series, several models have explored the integration of frequency embedding of timestamps for regression purposes. Taylor and Letham (2018) approached the time series as a continuous function of time using a general additive model (Hastie, 2017). They represented seasonality components as learnable Fourier series while explicitly specifying the ground truth seasonalities (e.g., weekly, monthly). Similarly, Hyndman and Athanasopoulos (2018) proposed embedding the timestamp t with the ground truth frequencies and applying a regression model to predict the series value at the timestamp t. Both methods rely on the explicit specification of seasonalities and are tailored for purely univariate time series, where information is not shared between samples.

In contrast, other models, such as TimeFlow or DeepTime (Woo et al., 2022), based on deep learning techniques, offer more flexibility. These approaches can autonomously learn relevant frequencies and effectively share information between samples through backpropagation. This enables a more dynamic and adaptable approach to time series forecasting, particularly in scenarios with complex temporal patterns and inter-sample dependencies.

F.2 Experiments

Given the seasonal patterns observed in the *Solar*, *Electricity*, and *Traffic* datasets, an alternative to deep learning forecasting methods is to individually regress a timestamp embedding on the corresponding value using a robust regressor. This approach exploits the inherent periodicity in the data, using timestamp embeddings to capture temporal dependencies and accurately predict the target values.

Baselines. We compare TimeFlow against two regression baselines:

- **TimeFlow frequencies embedding + XGBoost**: This baseline uses the same frequency embedding as TimeFlow and applies an XGBoost regressor (Chen and Guestrin, 2016) on top. The aim is to assess whether the XGBoost regressor can effectively identify the correct frequencies, filter out irrelevant ones, and establish the appropriate mapping between timestamps and values.
- **XGB Explicit Seasonal encoding + XGboost**: in this baseline, we give *explicitly* the right frequencies of each datasets to the model. It is important to highlight that **this method uses information that other baselines don't**. For instance for the *Traffic* dataset, the Explicit Seasonal encoding is $\gamma(t) = \left(t, \cos\left(\frac{2\pi t}{24}\right), \sin\left(\frac{2\pi t}{24}\right), \cos\left(\frac{2\pi t}{24\times 7}\right), \sin\left(\frac{2\pi t}{24\times 7}\right)\right)$. It allows to explicitly integers trend, daily frequencies and weekly frequencies. We apply the same type of frequencies embedding for *Solar* and *Electricity* with the appropriate seasonalities.

Experimental Setup. For each dataset and sample, we applied the frequency embedding individually and then trained an XGBoost regressor on each observed timestamp (the look-back window in forecasting and the observed grid in imputation). This approach results in one model per sample. The XGBoost regressor is configured with the following hyperparameters:

- n estimators: 500
- max depth: 4
- learning rate: 0.1
- lambda: [0.1, 1, 10]

The optimal regularization parameter lambda is determined through cross-validation. The imputation and forecasting results are presented in detail in Table 25 and Table 26, respectively.

Table 25: Mean MAE imputation results on the missing grid only. In the table, τ stands for the subsampling rate, i.e. the proportion of observed points considered for each samples. Bold results are best, underlined results are second best.

	τ	TimeFlow	TimeFlow frequencies embedding+ XGB	Explicit Seasonal encoding + XGB
Electricity	0.05	**0.324 ± 0.013**	0.834 ± 0.092	0.365 ± 0.051
	0.10	**0.250 ± 0.010**	0.761 ± 0.074	0.318 ± 0.049
	0.20	**0.225 ± 0.008**	0.632 ± 0.066	0.278 ± 0.044
	0.30	**0.212 ± 0.007**	0.536 ± 0.041	0.259 ± 0.048
	0.50	**0.194 ± 0.007**	0.418 ± 0.042	0.238 ± 0.022
Solar	0.05	**0.095 ± 0.015**	0.603 ± 0.035	0.234 ± 0.021
	0.10	**0.083 ± 0.015**	0.478 ± 0.024	0.190 ± 0.022
	0.20	**0.072 ± 0.015**	0.350 ± 0.022	0.150 ± 0.019
	0.30	**0.061 ± 0.012**	0.286 ± 0.018	0.134 ± 0.011
	0.50	**0.054 ± 0.013**	0.227 ± 0.015	0.123 ± 0.015
Traffic	0.05	**0.283 ± 0.016**	0.739 ± 0.140	0.344 ± 0.036
	0.10	**0.211 ± 0.012**	0.676 ± 0.129	0.290 ± 0.029
	0.20	**0.168 ± 0.006**	0.562 ± 0.108	0.245 ± 0.027
	0.30	**0.151 ± 0.007**	0.487 ± 0.095	0.223 ± 0.015
	0.50	**0.139 ± 0.007**	0.393 ± 0.083	0.198 ± 0.021
TimeFlow improvement		/	69.5 %	33.7 %

Imputation results. TimeFlow performs better than the other two methods. Although the second baseline explicitly incorporates ground truth frequencies and provides decent results, its inability to share information between samples leads to the loss of valuable insights that TimeFlow effectively exploits. In addition, the first baseline struggles to learn the correct frequencies and overfits observed data points, resulting in excessive high-frequency noise. As a result, its performance degrades significantly compared to the second baseline, where frequencies are explicitly provided. These findings underscore TimeFlow's ability to identify the underlying frequencies, and leverage shared information across samples to improve accuracy during imputation.

Table 26: Mean MAE forecast results for adjacent time windows averaged over different time windows. Each time, the model is trained on one time window and tested on the others (there are 2 windows for SolarH and 5 for Electricity and Traffic). H stands for the horizon. Bold results are best, and underlined results are second best

	H	TimeFlow	TimeFlow frequencies embedding + XGB	Explicit Seasonnal encoding + XGB
Electricity	96	**0.218 ± 0.017**	0.662 ± 0.102	0.282 ± 0.020
	192	**0.238 ± 0.012**	0.750 ± 0.128	0.279 ± 0.021
	336	**0.265 ± 0.036**	0.809 ± 0.136	0.294 ± 0.041
	720	**0.318 ± 0.073**	0.852 ± 0.144	0.357 ± 0.092
SolarH	96	**0.172 ± 0.017**	0.792 ± 0.062	0.244 ± 0.023
	192	**0.198 ± 0.010**	0.933 ± 0.055	0.236 ± 0.018
	336	**0.207 ± 0.019**	1.033 ± 0.052	0.229 ± 0.022
	720	**0.215 ± 0.016**	1.116 ± 0.057	0.262 ± 0.021
Traffic	96	**0.216 ± 0.033**	0.655 ± 0.156	0.288 ± 0.052
	192	**0.208 ± 0.021**	0.678 ± 0.139	0.246 ± 0.033
	336	**0.237 ± 0.040**	0.719 ± 0.143	0.262 ± 0.044
	720	**0.266 ± 0.048**	0.741 ± 0.140	0.288 ± 0.063
TimeFlow improvement		/	70.8 %	15.7 %

Forecasting results. TimeFlow also outperforms the other two baselines in forecasting. However, the improvement over the second baseline, where the correct frequencies are explicitly provided, is more modest compared to the gains observed in imputation. Nevertheless, the relative improvement achieved by TimeFlow remains significant (exceeding 15 %). Similar to the imputation scenario, the XGBoost baseline with Time-

Flow timestamps encoding, which attempts to learn the correct frequencies, fails to discern them accurately and introduces excessive high-frequency components.

G Latent space exploration

G.1 Latent space interpolation between two learned codes

It is interesting to understand how the latent space behaves between two learned codes, z_1 and z_2, which are representations of x_1 and x_2. We propose in Figure 14 to visualize how new $z_\lambda = \lambda z_1 + (1-\lambda) z_2$ are decoded in the time-series domain ($f_{\theta,h_w(z_\lambda)}(t)$ values).

Setting. We choose two z_1 and z_2 learned from the *Electricity* dataset and interpolate these two latent codes for $\lambda \in \{0.0, 0.1, 0.25, 0.50, 0.75, 0.9, 1.\}$.

Results. In Figure 14, we observe that the interpolation path between two codes yields a smooth transition in the time series domain. This suggests that the latent space is smooth and well-structured, and that the learned representations captured meaningful features of the time series, which could explain TimeFlow's generalization property.

G.2 TimeFlow sensitivity to modulations perturbation

In the preceding section, we observed the smoothness of the latent space. A crucial question arises: can we interpret each dimension of the latent space independently ?

Setting In this setting, we perturbed a specific dimension of the modulation (by adding Gaussian noise) for only one particular layer of the INR for the *Electricity* dataset. Then, we observe the difference in the time domain between the non-perturbated TimeFlow and the perturbated one. For instance in Figure 15, we add noise only for the third layer of the INR and the 50th channel.

Results. In Figure 15, we observed that adding a small perturbation added a smooth daily frequency pattern. In Figure 16, we observed that adding a small perturbation induces a bias that impacts the high frequencies but does not affect the low frequencies. Finally, in Figure 17, adding a small perturbation induces a very local and slight bias (the effect is almost null). In conclusion, the impact of the small perturbation depends on the channel and the layer, but it is hard to interpret each dimension independently.

G.3 Visualization of two code distributions in the latent space

Examining the behavior of the latent space at the instance level is of particular interest. It provides insights into how individual time series evolve within the latent space. However, exploring the latent space between two time series distributions is also crucial.

Setting. We propose encoding all samples (321 samples) from the *Electricity* dataset for two distinct time periods (each period is about 25 days $\approx$ 600 timestamps). This results in two distributions of latent codes, each representing different temporal support. Then, we employ Principal Component Analysis (PCA) to visualize these two latent code distributions in a two-dimensional space, as illustrated in Figure 18. This visualization allows us to explore the structural differences, similarities, and temporal variations in the latent space representations across the specified time intervals. In Figure 18a, the two compared time period are separated by approximately 3 months ($\approx$ 2000 timestamps). In Figure 18b, the two compared time period are separated by approximately 6 months ($\approx$ 4000 timestamps).

Results. Figure 18a shows that when the temporal periods are not too far from each other, the distributions of codes can largely overlap in the 2D visualization from the PCA. Conversely, as illustrated in Figure 18b, when the temporal periods are far from each other, the distributions of codes between the two periods become more distinct in the 2D visualization. This observation suggests that the proximity or disparity in temporal

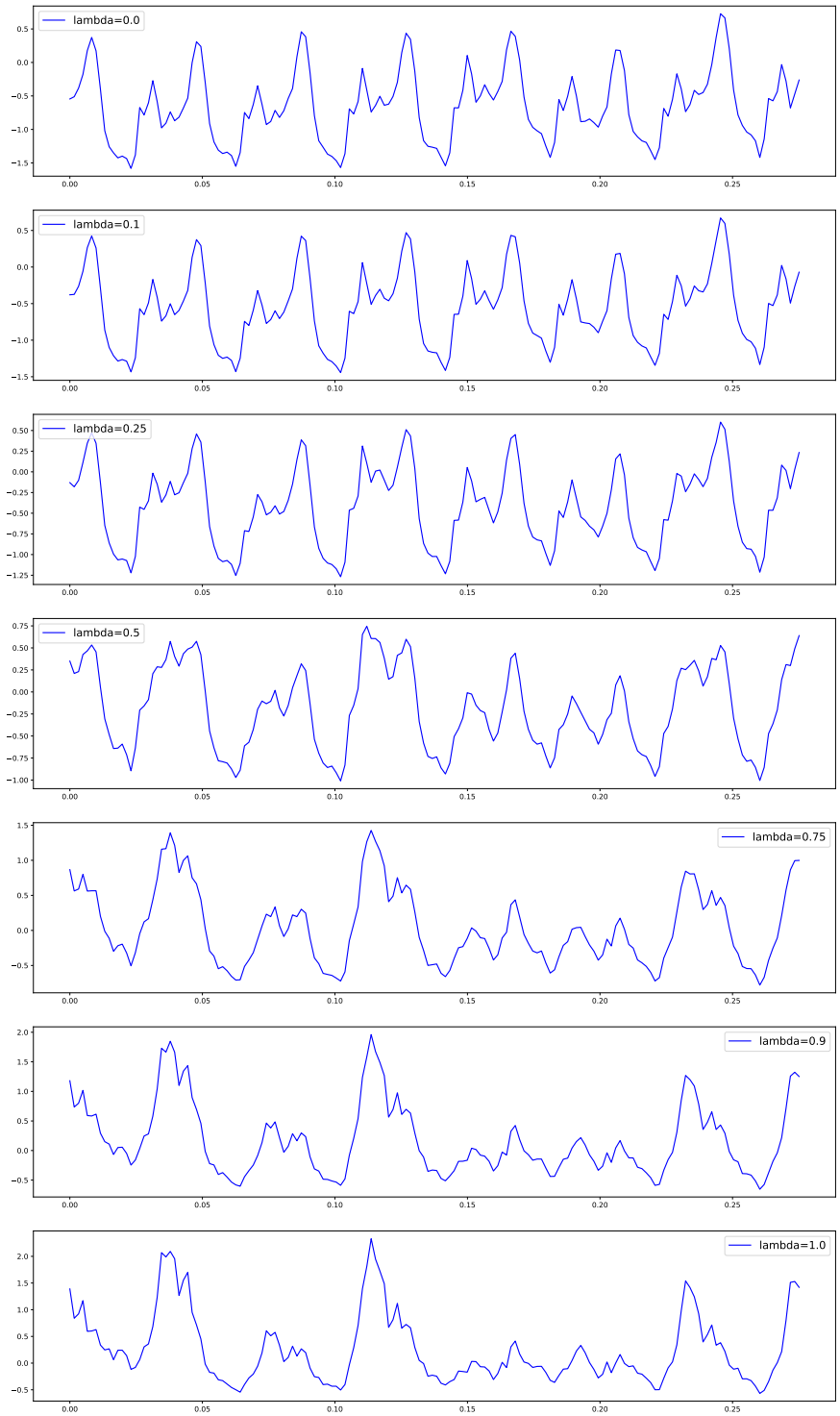


Figure 14: Visualization of the reconstructed time series for different linear interpolation of the two codes z_1 and z_2 learned from the *Electricity* dataset.

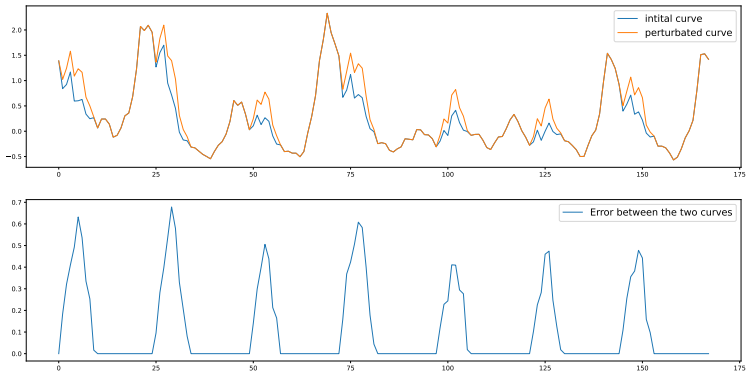


Figure 15: Effect of adding a small perturbation to the modulation shift of the third layer and 50th channel.

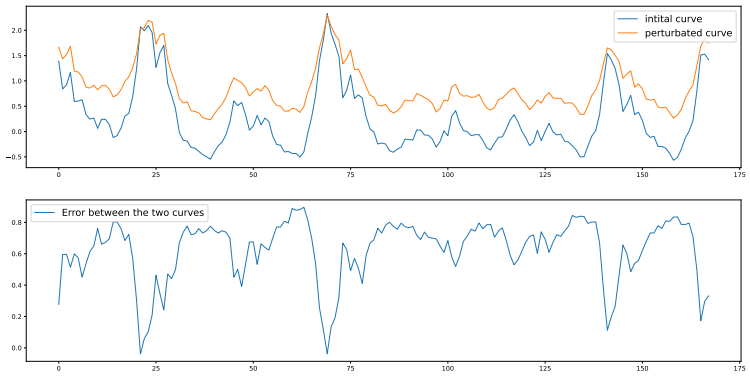


Figure 16: Effect of adding a small perturbation to the modulation shift of the third layer and 51th channel.

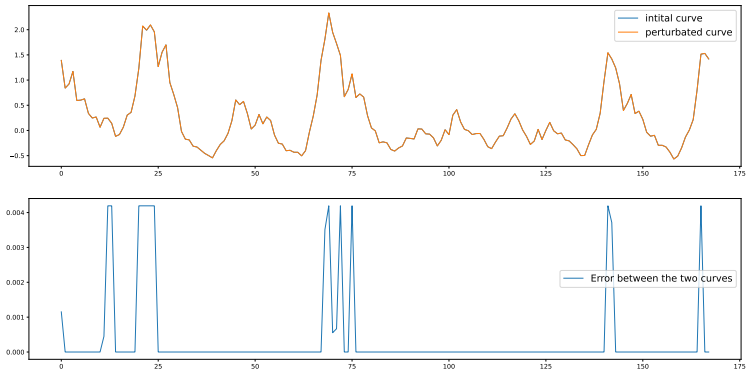
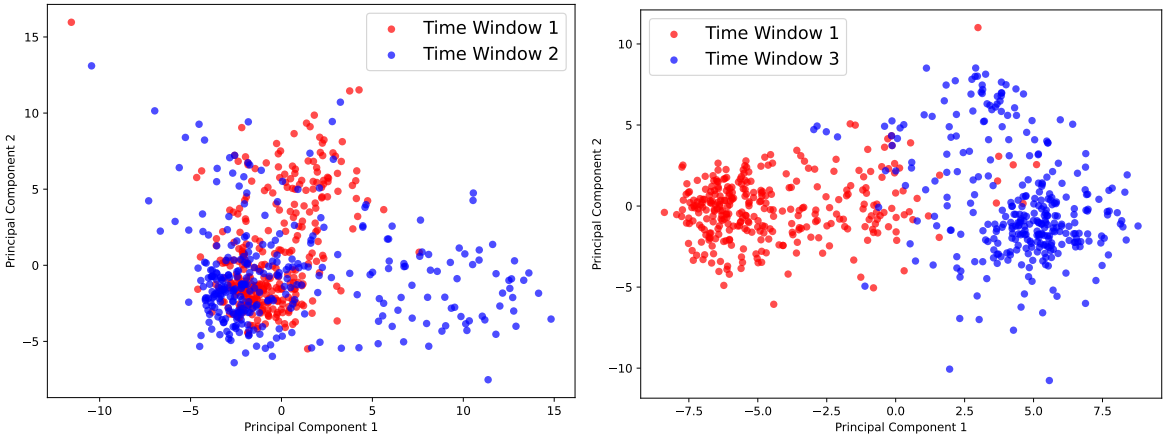


Figure 17: Effect of adding a small perturbation to the modulation shift of the fourth layer and 50th channel.

distribution shift influences the separability of latent representations in the 2D PCA space. However, this presumed separability in the latent space does not seem to significantly impact the generalization performance of TimeFlow across time, as evidenced by the results presented in Table 2 and Table 21. This suggests that our INR can handle relatively diverse code distributions.



(a) The temporal shift between the two codes distribution is 3 months.

(b) The temporal shift between the two codes distribution is 6 months.

Figure 18: Visualization of the two first PCA axes for two distributions of the latent codes (temporal distribution shift).

H The intuition behind the meta-learning optimization in time series forecasting

The concept of inner and outer loops is to be reformulated within the broader general framework of model-agnostic meta-learning (Finn et al., 2017), where the authors seek to enable rapid adaptation of the model to unseen tasks. In TimeFlow, we adapt the general idea of agnostic meta-learning to our tasks. We propose an efficient way to achieve this goal by splitting the parameters into two parts: context parameters (learned in the inner loop, responsible for the adaptive part of the model) and meta-parameters (or 'parameters shared across tasks') (learned in the outer loop, responsible for the generic part of the model).

In the context of time series forecasting. TimeFlow aims to have a subset of parameters that adapts to specific contextual factors (e.g., the look-back window of a particular sample) and another subset that performs the forecasting task according to this learned context (e.g., forecasting any point within the forecast horizon as well as within the look-back window). To achieve this, we seek to adjust the codes $z^{(j)}$ exclusively on the contexts (e.g., the look-back window) for each sample j, while the shared parameters between samples θ and w characterize a shared function capable of forecasting based on a given $z^{(j)}$. This function could be represented as $f_{\theta,w}(t, z(x^{(j)}))$. This concept entails adapting $z^{(j)}$ by sample j and training θ, w by batch.