Zero-Shot Forecasting and Neural Operators
Master Lab

Arwin Sadaghiani [1] **Jan Hardtke** [1] **Nadia Gharbi** [1]

Abstract

Time series forecasting plays a crucial role across domains, requiring accurate predictions of future horizons. This work introduces a zero-shot framework leveraging neural operators, consisting of a two-step architecture: a local embedding model (FIM-ℓ) that maps sparse time-series data to continuous function spaces, and a global prediction model (FIM) for forecasting. A synthetic dataset generated via Gaussian Processes with diverse kernels and noise was used for training and evaluation. Architectural enhancements, including positional encoding and cosine similarity loss, were explored to improve performance. Results emphasize the significance of robust data generation and real-world dataset integration to enhance generalization and applicability.

1. Introduction

Time series modeling plays a vital role in fields such as climate science (Waqas et al., 2024), medicine (Juang et al., 2017), finance (Sezer et al., 2020), and retail (Sousa et al., 2022). The objective is to capture temporal patterns for predictions of future time horizons. However, real-world time series data are often sparse and noisy, requiring interpolation to create a smooth, continuous representation. This assumes the data stem from an underlying hidden function, which we can approximate for tasks like forecasting.

In this report, we propose a machine learning approach using neural operators for time series forecasting in a zero-shot learning setting. Zero-shot learning enables models to generalize to new datasets, even beyond their training distribution, making it a promising method for building foundational models. As real-world data is often scarce, we generate multiple synthetic datasets by creating functions that capture trends and periodicity, combined with Gaussian noise to model observations of real-world phenomena.

The core idea of our method is to train a neural operator capable of mapping sparse time series observations into a smooth, continuous function space. This enables querying the predicted continuous function at arbitrary time points. The original concept is inspired by the FIM paper (Seifner et al., 2024), which leverages such neural operators for zero-shot imputation of observation gaps. Our approach extends this method by shifting the focus from imputing missing values to reconstructing a future time horizon based on observed past values.

The contributions of this work are as follows: (1) We create a synthetic dataset with 100K one-dimensional target functions sampled from a Gaussian Processes using multiple variants of kernels.

(2) Following the work of (Seifner et al., 2024), we utilize two models: FIM-ℓ, based on the work of DeepONet (Lu et al., 2021), and FIM (Seifner et al., 2024). The first model, FIM-ℓ, is responsible for learning local embeddings for localized windows of a given set of observations. These embeddings are then utilized by the second model, FIM, to predict the underlying function at future time horizons.

(3) Finally, we aim to refine these architectures by experimenting with different variations of the architecture and training objectives. These modifications are then evaluated on both our synthetic validation datasets and a real-world dataset.

2. Related Work

Statistical methods, such as ARIMA, have a long-standing history in time series forecasting. ARIMA, which stands for Autoregressive (AR), Integrated (I), and Moving Average (MA), combines these three components to model stationary time series data—where the mean and variance of the underlying process remain constant over time.

The autoregressive (AR) component predicts future values as a linear combination of previously observed values. The integrated (I) component addresses non-stationarity by applying differencing operations (of order d) to stabilize the mean and variance of the time series. Stationarity, a key assumption of ARIMA, is achieved through transformations such as differencing or logarithmic scaling. Finally, the moving average (MA) component incorporates past forecast errors into predictions, rather than relying solely on observed values.

These components are combined into the full ARIMA model, which is expressed as:

$$y'_t = c+\phi_1 y'_{t-1}+\cdots+\phi_p y'_{t-p}+\epsilon_t+\theta_1\epsilon_{t-1}+\cdots+\theta_q\epsilon_{t-q}, \quad (1)$$

where y'_t represents the differenced time series, ϕ and θ are the model coefficients, and ϵ_t is the forecast error. This formulation allows ARIMA to effectively capture the structure of stationary time series data while accounting for both observed values and forecast errors.

Although ARIMA performs well on univariate and stationary datasets, it faces limitations with non-linear patterns and high-dimensional data (Stellwagen & Tashman, 2013). To address these challenges, deep learning approaches have gained increasing attention (Stellwagen & Tashman, 2013). One recent advance is TimesFM (Das et al., 2024), a decoder-only foundation model designed for time series forecasting. Inspired by large language models, TimesFM is pre-trained on a diverse corpus of real-world and synthetic time series data, enabling it to capture complex temporal patterns across domains. It utilizes a decoder-style attention mechanism and input patching, similar to the approach used in vision transformers. TimesFM demonstrates strong zero-shot forecasting performance on both in-distribution and out-of-distribution (zero-shot) datasets, without requiring fine-tuning. Additionally, it is capable of handling varying forecasting horizons and time granularities.

Building on these advancements, this work investigates neural approaches, like TimesFM, to address challenges in time series forecasting.

3. Problem Definition

The goal of our project was to create a model which forecasts the future points of a time-series in a zero-shot manner. Concretely we want to build a model, that takes in $n \in \mathbb{N}$ windows of length $L \in \mathbb{N}$ of a time-series and then predicts window $n+1$. The model should also work in a zero-shot manner, meaning it should work without requiring any fine-tuning on the specific data it will be used on.

4. Methods

In this section we will look at the two main architectures that our model is based on. We will first look at DeepONet (Lu et al., 2021), which encodes our time-series windows and fits a function on the noisy time points, in Section 4.1 and afterwards at FIM (Seifner et al., 2024), which predicts the future encoding, in Section 4.2.

4.1. DeepONet (FIM-ℓ)

The Deep Operator Network (DeepONet) (Lu et al., 2021) is a neural network architecture designed to learn nonlinear operators, outperforming standard fully-connected networks. It comprises two sub-networks: a *branch-net* and a *trunk-net*. The *branch-net* encodes noisy observations at their respective time points into a latent representation, while the *trunk-net* encodes the query time point t for the hidden function into its own latent representation. Both sub-networks produce p-dimensional encodings, and the output function value at the query time point t is obtained by calculating the scalar product of these encodings. In this work, we consider the implementation of DeepONet, as proposed by FIM (Seifner et al., 2024), which we will refer to as FIM-ℓ.

4.2. FIM

FIM (Seifner et al., 2024) is a model originally designed for the interpolation of time series with temporal missing patterns. Given k embeddings $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{q-1}, \mathbf{h}_{q+1}, \ldots, \mathbf{h}_k$, where the embedding at position q is missing, the objective of FIM is to predict the missing embedding $\mathbf{h}_q$ based on the remaining embeddings.

The task of FIM is to predict the embedding for a given window q using the embeddings from other windows. In our work, we extended the core ideas of FIM to focus on forecasting the future embedding h_{k+1}.

To achieve this, FIM takes as input the embeddings $h_1, \ldots, h_k$ and their corresponding local scale embeddings $s_1, \ldots, s_k$. The scale embeddings are derived by capturing the normalization values used to standardize the time-series data for each window and passing these values through a multi-layer perceptron (MLP) to obtain the embeddings.

The embeddings h_i and s_i are concatenated for each window and processed through a Transformer followed by a summary network to compute the future embedding h_{k+1}. Finally, the embedding h_{k+1} is input into the trunk network of FIM-ℓ to generate a continuous function $f(t)$, enabling predictions for future time points t.

5. Synthetic Dataset

To train and test our models, we generated two synthetic datasets, each tailored to its respective model, consisting of time series.

For the DeepONet model, we constructed a dataset comprising $100{,}000$ random one-dimensional target interpolation functions $f(\tau)$, where $\tau \in [0, 1]$. Each function is defined on a uniform grid of 128 points and sampled from a Gaussian Process (GP) with a mean function of 0 and a Radial Basis Function (RBF) kernel given by

$$k(\tau, \tau') = \sigma^2 \exp\left(-\frac{\|\tau - \tau'\|^2}{2l^2}\right),$$

where σ^2 is the variance of the Gaussian Process, determining the magnitude of variations in $f(\tau)$, and l is the length scale, controlling the smoothness of the sampled functions.

The length scale is drawn randomly from a beta distribution and scaled by 0.1 to increase the frequency of the sampled functions f in our dataset.

Depending on whether we are creating a training or validation dataset, the values of α and β for the beta distributions are uniform selected from either $[1.0, 2.0, 5.0]$ or $[0.5, 3.25, 7.8]$, respectively. The variance σ^2 for the kernel is drawn uniformly from the interval $[0.5, 2]$. We then sample random observation grid points τ from our functions, which serve as inputs for our model. A random subset of points, between $\min = 50$ and $\max = 90$, is selected from the 128 grid points, and the remaining points are padded with 0's. Additionally, we create a binary mask for each observation grid to indicate which indices are observed versus unobserved. This mask is provided to the model along with the observed points.

Half of the observation points τ are sampled regularly, while the other half are sampled irregularly in time. To account for noise in real-world applications, Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.1)$ with mean 0 is added to the observations after normalizing the data.

The normalization process involves applying z-scoring to the function values $f(\tau)$ and min-max scaling to the time points τ. The normalized values are computed as follows

$$\hat{f}(\tau) = \frac{f(\tau) - \mu}{\sigma}, \quad \hat{\tau} = \frac{\tau - \tau^{\text{min}}}{\tau^{\text{max}} - \tau^{\text{min}}},$$

where μ and σ denote the mean and standard deviation of the function values $f(\tau)$, and τ^{min} and τ^{max} represent the minimum and maximum values of the observation points τ. This approach ensures that the function values $\hat{f}(\tau)$ are standardized, while the time points $\hat{\tau}$ are scaled to the interval $[0, 1]$.

For the FIM model, we created a dataset comprising $100{,}000$ random one-dimensional target functions $f(\tau)$ defined on the interval $[0, 1]$ and evaluated on a fine grid of 640 points. These functions were sampled from Gaussian Processes with a mean function of 0 and different kernels. The dataset distribution is composed as follows: periodic kernel k_p (30%), locally periodic kernel k_{lp} (30%), linear plus periodic kernel k_{lpp} (20%), and linear times periodic kernel k_{ltp} (20%).

The kernels are defined as:

$$\begin{aligned}
k_p(\tau, \tau') &= \sigma^2 \cdot \exp\left(-\frac{2\sin^2\left(\pi|\tau - \tau'|/p\right)}{l^2}\right), \\
k_{lp}(\tau, \tau') &= k_p(\tau, \tau') \cdot \exp\left(-\frac{(\tau - \tau')^2}{2l^2}\right), \\
k_{lpp}(\tau, \tau') &= k_p(\tau, \tau') + \sigma^2 \cdot (\tau \cdot \tau'^T), \\
k_{ltp}(\tau, \tau') &= k_p(\tau, \tau') \cdot \sigma^2 \cdot (\tau \cdot \tau'^T).
\end{aligned}$$

In addition to the length scale l and variance σ^2, the kernels include a periodicity parameter p. The length scale and periodicity are sampled from beta distributions. The parameters for these beta distributions are drawn either from $[1.0, 2.0, 5.0]$ for the training dataset or $[1, 1, 1]$ for the validation dataset.

The variance is drawn in the same manner as for the DeepONet dataset. We again sample random observation grid points τ from our functions, which serve as inputs for our model. We divide the observations into $K = 5$ windows, where the first 4 windows serve as the context, and the K-th window is used as the prediction ground truth.

The first 4 windows are constructed in the same manner as for the DeepONet dataset, except that the standard deviation of the added noise is now drawn from $\mathcal{N}(0, 0.025)$. One important detail is that, before normalizing each 128-point window in the same way as for the DeepONet dataset, we first normalize the entire function by applying z-scoring to the function values and min-max scaling to τ.

Inspired by (Seifner et al., 2024), we save the local statistics of each window $l \leq K$ into a scales array s_l, defined as:

$$\begin{aligned}
s_l = [\mu, \sigma, &\max_\tau f(\tau) - \min_\tau f(\tau), \\
&f(\tau^{\text{first}}), f(\tau^{\text{last}}), f(\tau^{\text{last}}) - f(\tau^{\text{first}}), \\
&\tau^{\text{first}}, \tau^{\text{last}}, \tau^{\text{last}} - \tau^{\text{first}}].
\end{aligned}$$

where τ represents the sampled observations, and f denotes the noisy function values.

6. Model architecture

Next, we proceed with a detailed description of the architectures for both the FIM-ℓ and FIM models. The FIM-ℓ model serves as an operator, aiming to learn the underlying function from noisy samples, while the FIM model is designed as a forecasting framework. Our description closely follows the original implementations of both architectures, as outlined in the FIM paper (Seifner et al., 2024).

6.1. FIM-ℓ

The primary aim of the FIM-ℓ model is to learn the underlying function $f(\tau)$ that has been augmented by noise to generate the observed time series (y_i, τ_i). The model should allow querying interpolated values of the underlying function $f(\tau)$ at arbitrary time points, including those not present in the observed data. We can therefore think of FIM-ℓ as a learned *neural interpolation operator* that maps the observed data into a continuous function space. To achieve this, we will leverage the ideas and architecture proposed by DeepONet (Lu et al., 2021). Given our noisy input sequence $(y_1, \tau_1), \ldots, (y_l, \tau_l)$, with observation values $y_i \in \mathbb{R}$ and ordered observation times $\tau_i \in \mathbb{R}^+$, as well as query points t_i, we define two feedforward neural network (FFN) embedding networks, ϕ_0^θ and ϕ_1^θ, to transform both the observed values and time points into an embedded representation:

$$\hat{y_i} = \phi_0^\theta(y_i), \quad \hat{t_i} = \phi_1^\theta(t_i).$$

We then proceed by concatenating both components to obtain the individual observation embeddings:

$$\mathbf{y_i^\theta} = \text{Concat}(\hat{y_i}, \hat{t_i}).$$

Following the work of DeepONet, we define a *branch net*-equivalent network consisting of a transformer-encoder network (Vaswani et al., 2023), denoted as ψ_0^θ, and a multilayer perceptron (MLP), denoted as ϕ_3^θ. Together, these form

$$\mathbf{u}^\theta = \phi_3^\theta\big(\psi_0^\theta(\mathbf{y_1^\theta}, \dots, \mathbf{y_l^\theta})\big).$$

Finally, to generate a sequence-length-agnostic embedding, we take $\mathbf{u}^\theta$ from the branch network and feed it into a Multi-Head Attention (Vaswani et al., 2023) summary block λ_0^θ, where $\mathbf{u}^\theta$ serves as the *keys* and *values*, and a *learnable* vector q_{θ^*} is used as the *query*. The attention calculation is defined as

$$\mathbf{h}^\theta = \text{softmax}\left(\frac{q_{\theta^*} K^\top}{\sqrt{d_k}}\right) V = \lambda_0^\theta(\mathbf{u}^\theta),$$

where $K = \mathbf{u}^\theta$ are the keys, $V = \mathbf{u}^\theta$ are the values, and d_k is the dimensionality of the keys.

Next, we define our *trunk net*-equivalent network. We begin by introducing a separate embedding network, ϕ_4^θ, for the query points t. Additionally, we define another MLP, ϕ_5^θ. The final trunk net output, $\mathbf{t}^\theta$, is then obtained as

$$\mathbf{t}^\theta = (\phi_5^\theta \circ \phi_4^\theta)(t).$$

To finally obtain the interpolated values of the learned underlying function at the query points t, we define a final MLP, ϕ_7^θ, such that

$$\mathbf{y}(t) = \phi_7^\theta\big(\text{Concat}(\mathbf{h}^\theta, \mathbf{t}^\theta)\big),$$

where $\mathbf{y}(t)$ represents the learned underlying function given our noisy observation values.

6.2. FIM

We now proceed by utilizing the learned representations $\mathbf{h}^\theta$ of each local function window to predict the values of the time series at arbitrary points within the next, previously unseen window. Starting from the beginning, we receive a noisy time sequence $(y_1, \tau_1), \dots, (y_l, \tau_l)$. We then split these values into $K = 5$ windows, such that for window S_j, we have

$$S_{ji} = (y_{\alpha+i}, \tau_{\alpha+i}), \quad \alpha = \sum_{l=1}^{j-1} w_l,$$

where w_l is the number of observations in window $l \leq K-1$. Additionally, for each of these windows, we construct their local scale characteristics $s_l \in \mathbb{R}^9$ 5, which are fed into an embedding layer σ_0^ω, defined as

$$\hat{s_l} = \sigma_0^\omega(s_l).$$

We then pass each window of observations into the pretrained embedding layers of the FIM-ℓ model. Specifically, we define:

$$\mathbf{y}_i^j = \text{Concat}\big(\phi_0^\theta((S_{ji})_1), \phi_1^\theta((S_{ji})_2)\big), \quad \text{j} \leq K-1, i \leq w_j.$$

We proceed by passing these $\mathbf{y}^j$ into the branch network of the FIM-ℓ, resulting in

$$\mathbf{h}_j = (\lambda_0^\theta \circ \phi_3^\theta \circ \psi_0^\theta)\big(\mathbf{y}_1^j, \dots, \mathbf{y}_{w_j}^j\big).$$

After extracting the local embeddings for each of the $K-1$ windows, we proceed to reconstruct the K-th window. To achieve this, we first concatenate each local-scale embedding $\hat{s}_j$ with the observation embeddings and feed them into a Transformer encoder block ψ_1^ω. This is again followed by an attention-based summary network λ_1^ω, which generates the final embedding for the K-th window

$$\mathbf{h}_K^* = (\eta_0^\omega \circ \lambda_1^\omega \circ \psi_1^\omega)\Big((\mathbf{h}_1, \hat{s}_1), \dots, \eta_0^\omega(\mathbf{h}_{K-1}, \hat{s}_{K-1})\Big).$$

Due to the concatenation of the observation and scale embeddings, the feature dimension is now doubled compared to the original embedding size. However, the frozen projection network of FIM-ℓ expects inputs in the original embedding dimension. To address this, we utilize an extractor network η_0^σ, which transforms the output of the summary network λ_1^ω back to the dimension expected by the FIM-ℓ projection layer.

To generate the final predictions for the K-th window, we utilize the embedding and trunk networks of the pretrained FIM-ℓ to predict the function values at the query points t. This is expressed as

$$\mathbf{y}(t) = \phi_7^\theta\big(\text{Concat}(\mathbf{h}_K^*, (\phi_5^\theta \circ \phi_4^\theta)(t))\big).$$

7. Model Training

In this section, we provide the necessary information to ensure the reproducibility of our work. Specifically, we outline the detailed structure of the aforementioned MLPs and discuss all relevant hyperparameters for both the model and the optimization algorithms.

7.1. FIM-ℓ Training

The implementation of our previously defined FIM-ℓ architecture is described in 7.1. Here, we use d_{model} to denote the embedding dimension. In our setting, we set $d_{\text{model}} = 256$ and $n_{\text{heads}} = 4$. Since *LeakyReLU* is shift-invariant, the bias term in the linear layer can be omitted if it is followed by a *LayerNorm*, as the *LayerNorm* neutralizes any bias introduced by the preceding layer. We train the model with a batch size of 128, using the *AdamW* optimizer with the following hyperparameters: β-values $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and a weight decay of 0.01. The training is performed for 20 epochs, which took approximately one hour. Additionally, we employ an *Inverse Square Root Learning Rate* (*InverseSquareRootLR*)(**?**) scheduling strategy, with 100 warm-up steps, an initial learning rate of 10^{-4}, and a minimum learning rate of 10^{-5}.

To save memory and computational resources, we utilize the PyTorch *Automatic Mixed Precision* package, which trains the model in mixed precision. Specifically, it selects half-precision data types (*bfloat16* in our case) for operations it deems suitable. This approach enables the model to leverage the highly optimized NVIDIA Tensor Cores, maximizing performance during matrix operations.

For our loss computations, we use the standard *Mean Squared Error* (MSE) between the predicted outputs of the model and the precomputed ground truth. Before performing the optimizer update step, we apply gradient clipping to ensure that the gradient norm does not exceed the length of a unit vector. This stabilizes training by limiting the size of each gradient step during optimization.

We then provide the model with the noisy observation sequence, observation time points, query points, and the branch mask. The branch mask is then utilized by both the Transformer encoder ψ_0^θ and the summary network λ_0^θ as the padding mask.

7.2. FIM Training

The implementation of the FIM network that we defined is detailed in Table 7.2. We set $d_{\text{model}} = 256$ and $n_{\text{heads}} = 8$. Regarding the optimizer and learning rate strategy, we use the same settings as described previously, along with automatic mixed precision training. The loss function remains the *Mean Squared Error* (MSE), and the gradients are clipped to ensure their norm does not exceed the length of a unit vector. We again provide the model with the noisy observation sequence, observation time points, query points, and the branch mask. However, instead of treating a single window as one data point, each example now comprises all local windows of the global function. We then continue training for 70 epochs, which takes approximately 4 hours. All model training was conducted on an NVIDIA RTX 3070 GPU, equipped with 8GB of GDDR6 VRAM.

Component	Details
Branch Embedding ϕ_0^θ	Linear(1, d_{model})
Branch Embedding ϕ_1^θ	Linear(1, d_{model})
Trunk Embedding ϕ_4^θ	Linear(1, d_{model})
Branch Encoder Input	Concatenate embeddings of y and t
Branch Encoder ψ_0^θ	Transformer Encoder (6 layers, $2d_{\text{model}}$, n_{heads})
Branch MLP ϕ_3^θ	Linear($2d_{\text{model}} \to d_{\text{model}}$), LeakyReLU, LayerNorm
Learnable Query	Parameter tensor of shape (1, d_{model})
Branch Attention λ_0^θ	Multihead Attention (d_{model}, heads=1)
Trunk MLP ϕ_5^θ	4x Linear($d_{\text{model}} \to d_{\text{model}}$), LeakyReLU, LayerNorm
Combine Outputs	Concatenate outputs of Branch Attention and Trunk MLP
Final Projection ϕ_7^θ	5x Linear ($2d_{\text{model}} \to 1$), LeakyReLU, LayerNorm

Table 1. FIM-ℓ architecture implementation

8. Experiments

In this section, we discuss additional experiments conducted on the model's architecture and the construction of the loss function to achieve higher prediction accuracy. The impact of each of these approaches will be presented later in Section 9.

8.1. Learned Positional Encoding for FIM-ℓ Embeddings

In this approach, we hypothesize that the model may benefit from additional positional encoding for the attention-based summary network λ_1^ω. We define

$$\mathbf{p} = \psi_1^\omega\Big((\mathbf{h}_1, \hat{s}_1), \ldots, \eta_0^\omega(\mathbf{h}_{K-1}, \hat{s}_{K-1})\Big), \quad \mathbf{p} \in \mathbb{R}^{(K-1)\times d_{\text{model}}}.$$

as the output from the FIM transformer encoder network. In our standard implementation, $\mathbf{p}$ is passed to the summary attention network λ_1^ω, which, by default, lacks a sense of order among these embeddings, aside from the local scale

Component	Details
Pretrained FIM-ℓ	$n_{\text{heads-fim-}\ell}$, d_{model} (Frozen)
Local Scale Embedding σ_0^ω	Linear(9, d_{model})
Combine Outputs	Concatenate outputs of Pretrained FIM-ℓ and Local Scale Embedding
Transformer Encoder ψ_1^ω	8 layers, $2d_{\text{model}}$, n_{heads}
Learnable Query	Parameter tensor of shape (1, $2d_{\text{model}}$)
Summary Attention λ_1^ω	Multihead Attention ($2d_{\text{model}}$, n_{heads})
Extractor Network η_0^ω	4x Linear ($2d_{\text{model}} \rightarrow d_{\text{model}}$), LeakyReLU, LayerNorm
Trunk Embedding ϕ_4^θ	Reused from FIM-ℓ (Frozen)
Trunk MLP ϕ_5^θ	Reused from FIM-ℓ (Frozen)
Combine Outputs	Concatenate outputs of Trunk Network and Summary Network
Final Projection ϕ_7^θ	Reused from FIM-ℓ (Frozen): 5x Linear ($2d_{\text{model}} \rightarrow 1$), LeakyReLU, LayerNorm

Table 2. FIM Architecture Overview

statistics of each window. We hypothesize that providing positional information may help this layer better identify the order of embeddings, enabling more informed predictions of the K-th embedding.
To address this, we introduce an additional parameter vector $\mathbf{z} \in \mathbb{R}^{(K-1)\times d_{\text{model}}}$, which serves as a learnable positional encoding. We add $\mathbf{z}$ elementwise to $\mathbf{p}$ before passing the result to λ_1^ω, as follows

$$\mathbf{h}_K^* = (\eta_0^\omega \circ \lambda_1^\omega)\Big(\mathbf{p} + \mathbf{z}\Big).$$

8.2. Similarity loss between FIM-ℓ ground truth and predicted embeddings.

Another strategy is the usage of the ground truth $\mathbf{h}_K$ for enabling more accurate predictions. The idea is to align the predicted $\mathbf{h}_K^*$ and the actual ground truth $\mathbf{h}_K$ such that they are as close as possible to each other. We can do this by adding an additional term to the model's loss function, which incentivizes it to form predictions $\mathbf{h}_K^*$ that are close to the FIM-ℓ predicted $\mathbf{h}_K$. A suitable metric for this is the *cosine similarity*, defined as

$$\text{CosSim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}, \quad \mathbf{a}, \mathbf{b} \in \mathbb{R}^n.$$

$$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n \setminus \{\mathbf{0}\}, \quad \text{CosSim}(\mathbf{a}, \mathbf{b}) \in [-1, 1].$$

Geometrically, the *cosine similarity* represents the cosine of the angle between two vectors. Specifically, $\text{CosSim}(\mathbf{a}, \mathbf{b}) = 0$ indicates orthogonality, while $\text{CosSim}(\mathbf{a}, \mathbf{b}) = 1$ and $\text{CosSim}(\mathbf{a}, \mathbf{b}) = -1$ correspond to vectors pointing in the same and opposite directions, respectively.
To give the network a better chance of transforming $\mathbf{h}_K^*$, we introduce an additional 4-layer MLP called λ_2^ω. We then calculate the cosine similarity (CosSim) between the output of λ_2^ω and $\mathbf{h}_K$.
We now define our new loss function $\mathcal{L}(y, \hat{y}, \mathbf{h}_K^*, \mathbf{h}_K)$, which incorporates both prediction accuracy and alignment between the predicted and ground-truth representations. The loss function is given by

$$\mathcal{L}(y, \hat{y}, \mathbf{h}_K^*, \mathbf{h}_K) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \beta \cdot |\text{CosSim}(\lambda_2^\omega(\mathbf{h}_K^*), \mathbf{h}_K) - 1|.$$

where y denotes the ground truth values of the target variable, and $\hat{y}$ represents the corresponding predicted values by the model. We choose $\beta = 0.2$ to ensure that the optimization does not focus too aggressively on aligning $\mathbf{h}_K^*$ and $\mathbf{h}_K$.

9. Results

We will now discuss our findings regarding both the baseline architecture and the performance of our additional experiments. For evaluation, we utilize both our validation set and the *ETTh1* dataset, which comprises real-world time series data from multiple domains.

9.1. Findings on Validation Set

On our validation set, we evaluate the performance of our models with the *Mean Absolute Error* (MAE) defined as

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{1}^{n} |y_i - \hat{y}_i|.$$

In Table 3, we present the results on our validation set for multiple instances of our model. We abbreviate our Standard model with the capital letter S, as well as PE for positional encoding and CSL for cosine-similarity loss. Unfortunately, as shown in the table, neither the addition of positional encoding to the FIM-ℓ embeddings in the summary network nor the introduction of the cosine loss yielded any significant benefit. However, we observed a slightly lower training loss for the PE variant.

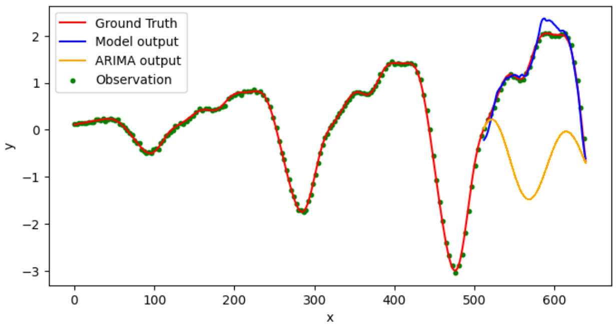

Figure 1. Example of good performance on a validation example exhibiting a clear pattern.

Looking at the plots of the predictions generated by our model on the validation set, we observed that the model performs well on data with high periodicity (see Figure 1). Since the functions and the parameters for the Gaussian process are sampled randomly, it is possible to encounter functions with less clear patterns (see Figure 2). These functions pose a challenge for the model, as it cannot anticipate their behavior in advance. This results in the model learning the average course of such functions. We postulate that this behavior negatively impacts the model's performance on functions where distinct patterns are present. It's therefore crucial to have checks in place to ensure that the model is only trained on data that exhibits clear patterns or aligns with the desired characteristics of the target task. This may help prevent the model from overfitting to noise or learning irrelevant trends.

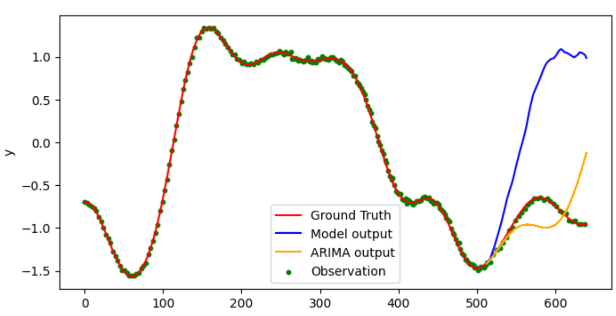

Figure 2. Example of bad performance on a validation example a less clear pattern.

9.2. Findings on ETTh1

ETTh1 is a time series dataset containing data from various domains, such as oil temperature and electrical charge. We evaluate our model with a time horizon of 96, which corresponds to capturing 480 data points. These are split into $K = 5$ windows, with the task being to predict the 5th window. The five windows are then shifted by one to the right and evaluated again. The same procedure is applied for the time horizon of 192. We evaluate our different

	S	PE	PE + CSL	TimesFM
Validation	0.576	0.579	0.585	-
ETTh1 - 96	0.966	0.968	0.928	0.43*
ETTh1 - 192	1.0904	1.089	1.043	0.43*

Table 3. MAE results for three model variants and TimesFM on two datasets.
* TimesFM (Das et al., 2024) only reports the average score over both time horizons. Additionally, they use only 4 datasets from ETTh1, which they do not disclose.

model variants on the entire ETTh1 dataset in a zero-shot manner, meaning that we evaluate on all sub-datasets contained within ETTh1 and calculate the average performance. Additionally, we compare our model to the one proposed by TimesFM (Das et al., 2024), using their accuracy score as a benchmark. As shown in Table 3, positional encoding does not appear to offer a significant benefit over the standard model. However, the PE + CSL approach provides slightly better performance. We also observe that the zero-shot performance of TimesFM on ETTh1-96 is much better than that of our current model. It is important to note, however, that TimesFM has been trained on a variety of data, both real-world and synthetic, and is approximately an order of magnitude larger in terms of parameter count. We observe the same ordering of models for a horizon window of 192, where the PE + CSL version performs slightly better than the vanilla and PE versions. We additionally provide plots showing our model's performance on the live dataset provided by our instructors (see Appendix A).

Finally, we present two examples from the ETTh1 dataset using our model to showcase both well-performing and poorly-performing cases. In Figure 3, the model successfully predicts the horizon, as the data follows a very predictable pattern. However, in Figure 4, we observe that the model struggles to accurately predict the horizon probably due to the lack of a very clear pattern in the data.

This suggests that our networks would likely benefit significantly from training on more diverse and real-world datasets, allowing them to learn patterns that are more subtle than those generated by a Gaussian process with a periodic kernel.

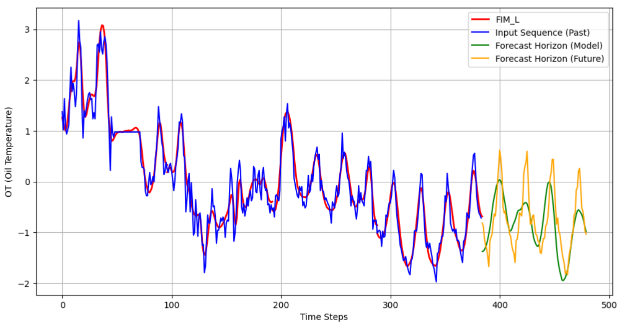

Figure 3. Example of good performance on the oil temperature dataset with horizon $= 96$, using the PE + CSL model version.

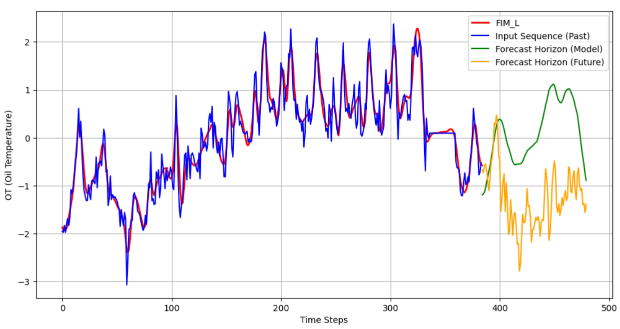

Figure 4. Example of bad performance on the oil temperature dataset with horizon $= 96$, using the PE + CSL model version.

10. Conclusion

In this work, we presented a novel approach for zero-shot forecasting by integrating the concepts of DeepONet and the FIM architecture to predict future horizons in time series data. To facilitate effective training, we constructed a synthetic dataset comprising randomly sampled periodic functions augmented with normally distributed noise.

To advance our architecture further, we experimented with several modifications, including the addition of positional encoding for local function embeddings and the introduction of an auxiliary loss function aimed at aligning the predicted embeddings of future horizons with their corresponding ground truth embeddings.

We evaluated our approaches on both a validation set and the *ETTh1* dataset. While the incorporation of positional encoding did not yield a significant performance improvement, the cosine similarity loss variant demonstrated a modest enhancement in predictive accuracy.

Our findings highlight the importance of carefully constructing datasets that exhibit genuine patterns to prevent the model from overfitting to noise. Additionally, integrating real-world datasets into the training process, as done by TimesFM, is essential for enhancing the model's applicability and performance in practical scenarios.

References

Das, A., Kong, W., Sen, R., and Zhou, Y. A decoder-only foundation model for time-series forecasting, 2024. URL https://arxiv.org/abs/2310.10688.

Juang, W.-C., Huang, S.-J., Huang, F.-D., Cheng, P.-W., and Wann, S.-R. Application of time series analysis in modelling and forecasting emergency department visits in a medical centre in southern taiwan. *BMJ Open*, 7(11), 2017. ISSN 2044-6055. doi: 10.1136/bmjopen-2017-018628. URL https://bmjopen.bmj.com/content/7/11/e018628.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://dx.doi.org/10.1038/s42256-021-00302-5.

Seifner, P., Cvejoski, K., Körner, A., and Sánchez, R. J. Zero-shot imputation with foundation inference models for dynamical systems. In *Submitted to The Thirteenth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NPSZ7V1CCY. under review.

Sezer, O. B., Gudelek, M. U., and Ozbayoglu, A. M. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181, 2020. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2020.106181. URL https://www.sciencedirect.com/science/article/pii/S1568494620301216.

Sousa, M., Tomé, A. M., and Moreira, J. Long-term forecasting of hourly retail customer flow on intermittent time series with multiple seasonality. *Data Science and Management*, 5(3):137–148, 2022. ISSN 2666-7649. doi: https://doi.org/10.1016/j.dsm.2022.07.002. URL https://www.sciencedirect.com/science/article/pii/S2666764922000273.

Stellwagen, E. and Tashman, L. Arima: The models of box and jenkins. *Foresight: The International Journal of Applied Forecasting*, (30), 2013.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Waqas, M., Humphries, U. W., and Hlaing, P. T. Time series trend analysis and forecasting of climate variability using deep learning in thailand. *Results in Engineering*, 24:102997, 2024. ISSN 2590-1230. doi: https://doi.org/10.1016/j.rineng.2024.102997. URL https://www.sciencedirect.com/science/article/pii/S2590123024012520.

A. Appendix

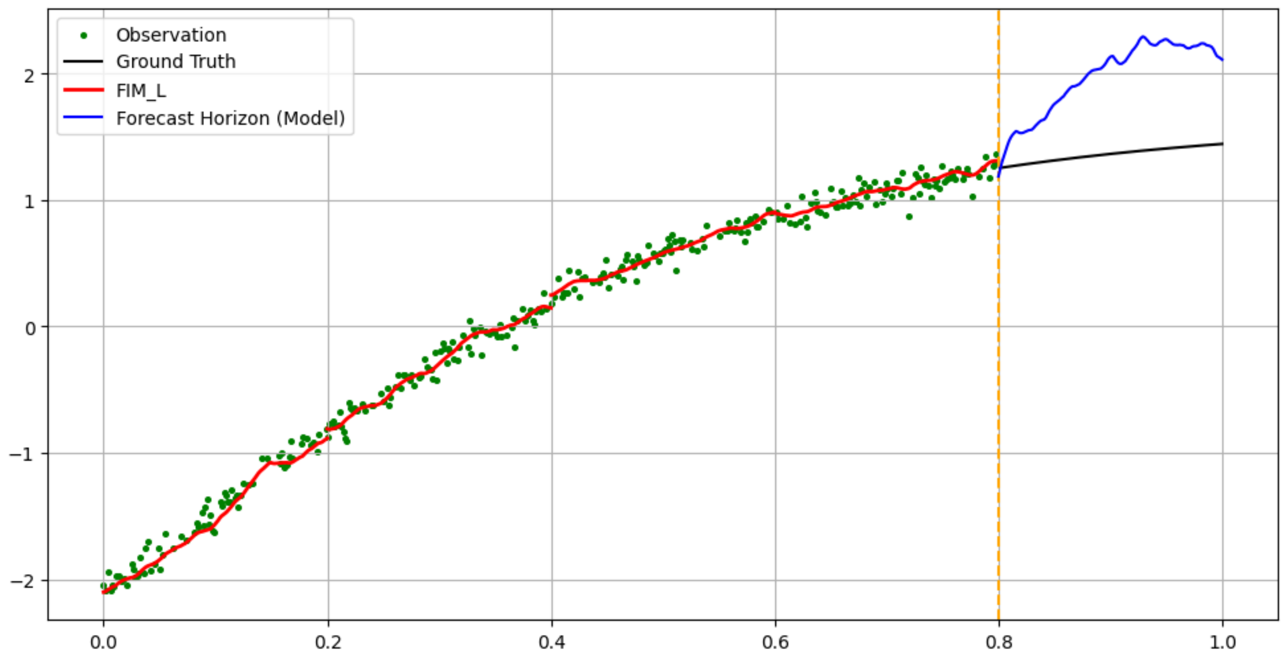

Figure 5. Example on the live dataset with horizon $= 128$, using the PE + CSL model version.

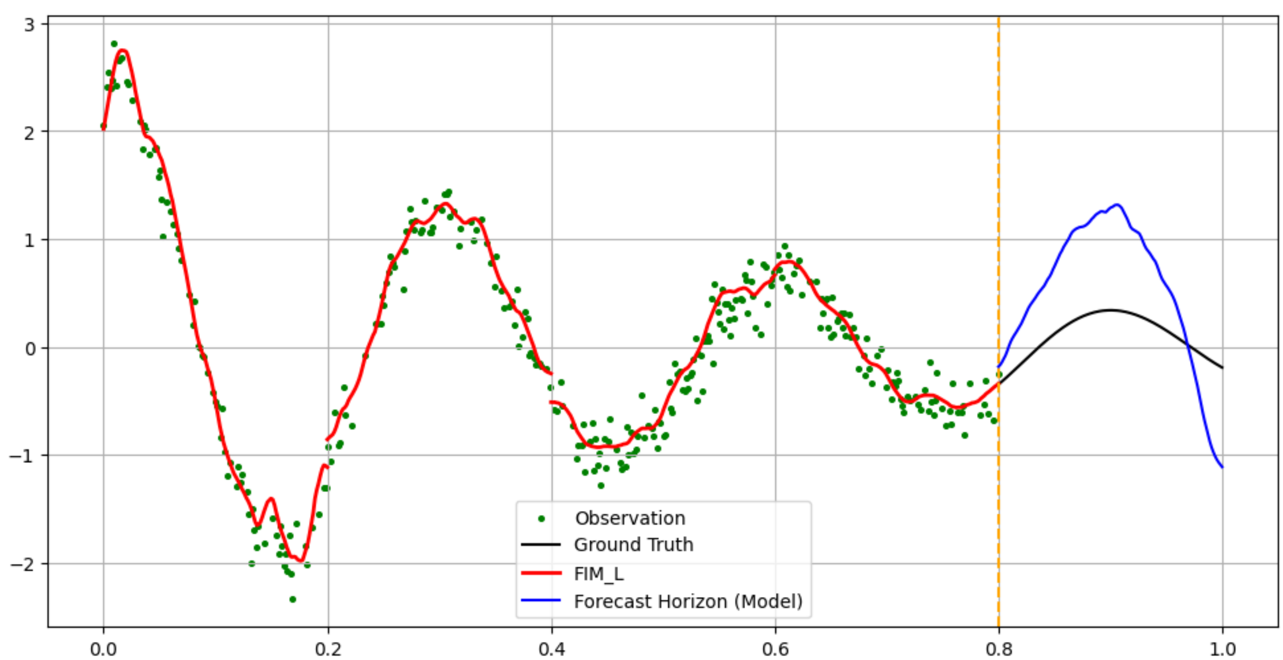

Figure 6. Example on the live dataset with horizon $= 128$, using the PE + CSL model version.

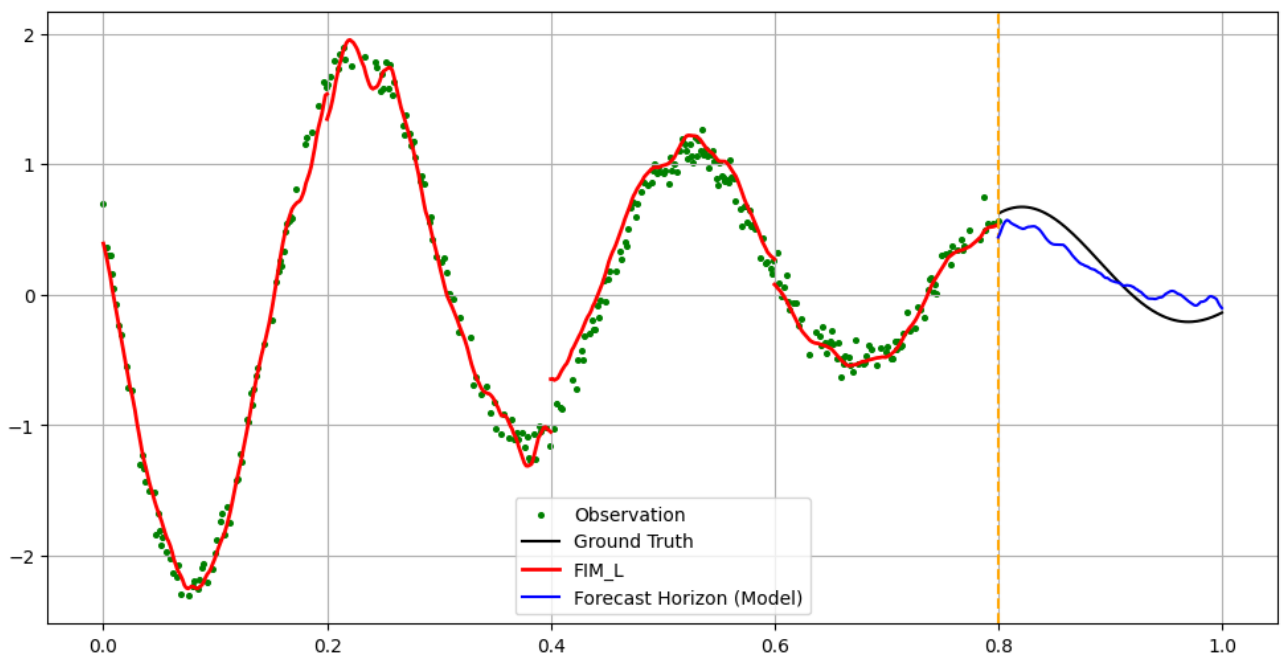

Figure 7. Example on the live dataset with horizon $= 128$, using the PE + CSL model version.

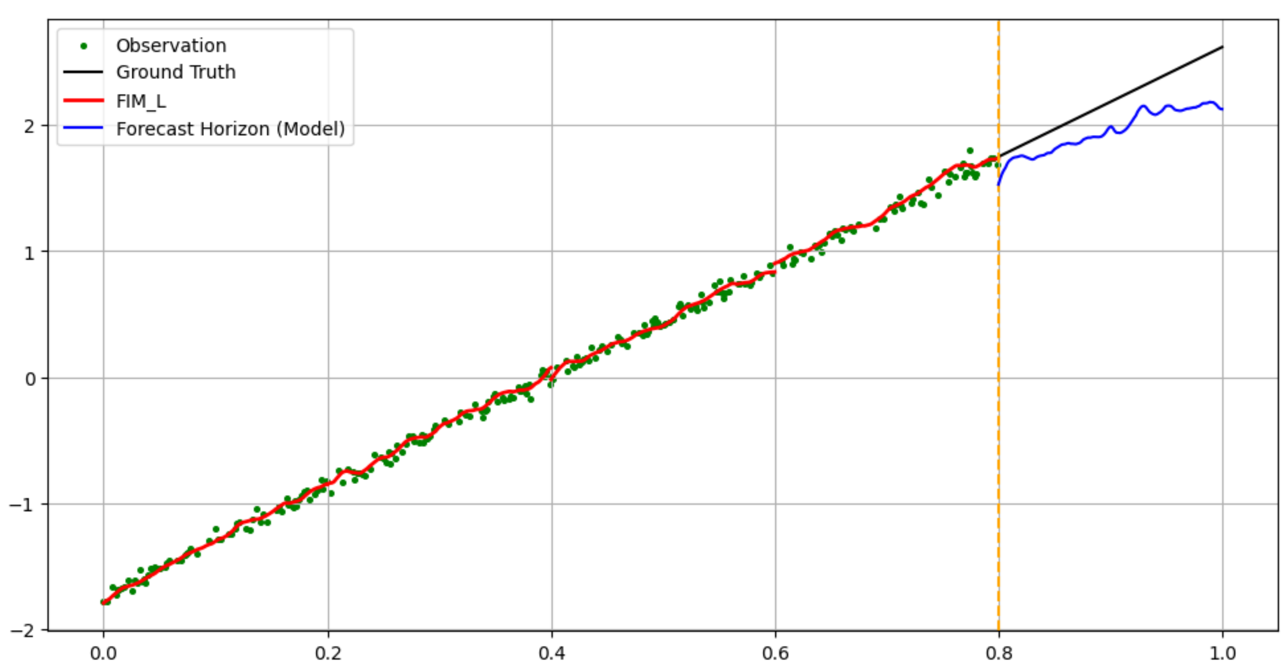

Figure 8. Example on the live dataset with horizon $= 128$, using the PE + CSL model version.

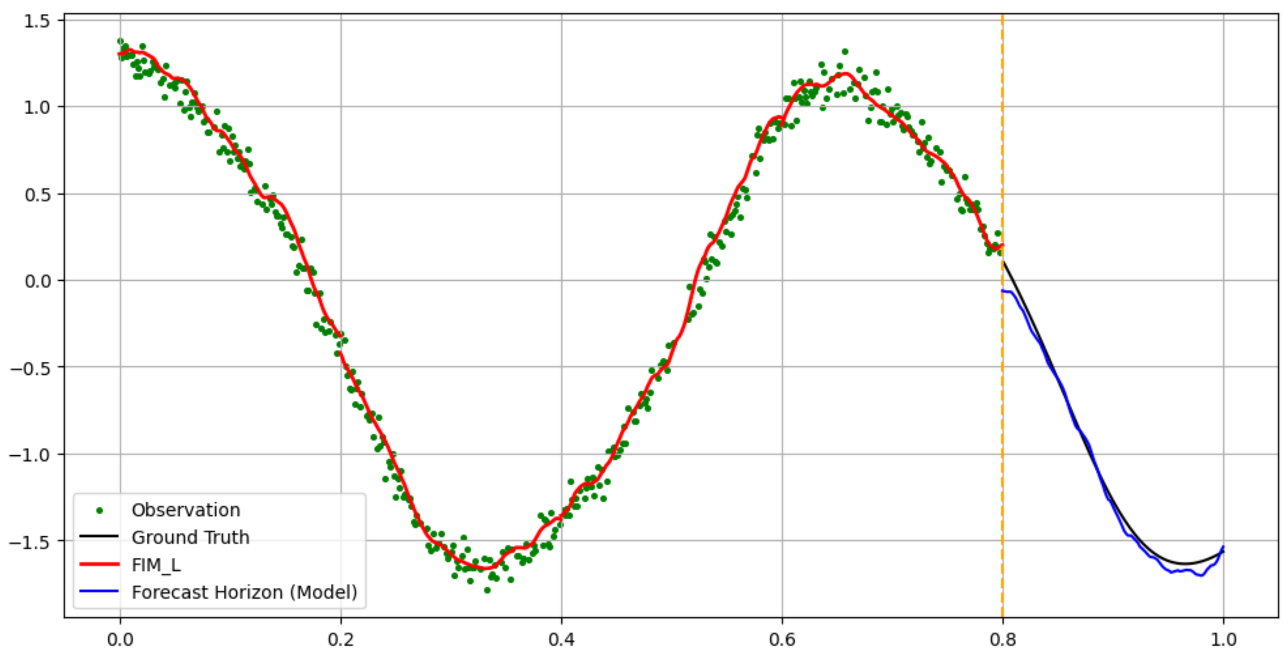

Figure 9. Example on the live dataset with horizon $= 128$, using the PE + CSL model version.