TimePFN: Effective Multivariate Time Series Forecasting with Synthetic Data

Ege Onur Taga M. Emrullah Ildiz Samet Oymak

University of Michigan, Ann Arbor
{egetaga,eildiz,oymak}@umich.edu

Abstract

The diversity of time series applications and scarcity of domain-specific data highlight the need for time-series models with strong few-shot learning capabilities. In this work, we propose a novel training scheme and a transformer-based architecture, collectively referred to as *TimePFN*, for multivariate time-series (MTS) forecasting. *TimePFN* is based on the concept of Prior-data Fitted Networks (PFN), which aims to approximate Bayesian inference. Our approach consists of (1) generating synthetic MTS data through diverse Gaussian process kernels and the linear coregionalization method, and (2) a novel MTS architecture capable of utilizing both temporal and cross-channel dependencies across all input patches. We evaluate *TimePFN* on several benchmark datasets and demonstrate that it outperforms the existing state-of-the-art models for MTS forecasting in both zero-shot and few-shot settings. Notably, fine-tuning *TimePFN* with as few as 500 data points nearly matches full dataset training error, and even 50 data points yield competitive results. We also find that *TimePFN* exhibits strong univariate forecasting performance, attesting to its generalization ability. Overall, this work unlocks the power of synthetic data priors for MTS forecasting and facilitates strong zero- and few-shot forecasting performance.

1 Introduction

Natural language processing has achieved significant success due to advances in neural architectures and data pipelines, leading to models with strong zero-shot and few-shot learning abilities. Inspired by this, researchers are exploring these methods for time series forecasting, focusing on transformer architectures [28, 25, 29, 27]. However, simple linear models often outperform transformers in time-series tasks due to data heterogeneity and naive tokenization methods [26]. As improvements, PatchTST does forecasting by patching the data [19], while iTransformer represents each channel as a single token for multivariate forecasting [16].

In this work, we take a data-centric approach to multivariate time series (MTS) forecasting, addressing the limitations of current models that struggle with small or perhaps out-of-distribution datasets. Our method, *TimePFN*, introduces two novelties: generating diverse, large-scale synthetic MTS data with inter- and intra-channel dependencies, and creating an architecture that extracts time series features from this data, enabling transfer learning to tasks with varying channel numbers. Our model achieves state-of-the-art zero-shot and few-shot accuracy on benchmarks, outperforming alternatives when fine-tuned on small datasets.

Key contributions include: **(i)** a new method for generating synthetic MTS data using Gaussian processes, **(ii)** an architecture that includes channel mixing and a convolutional embedding module, building upon PatchTST, **(iii)** the first multivariate time-series PFN model, and **(iv)** strong univariate forecasting performance, highlighting the flexibility and generalization of our approach.

NeurIPS 2024 Workshop on Time Series in the Age of Large Models.

2 Related Work

Transformers [23] have revolutionized NLP, boosting zero-shot and few-shot capabilities in language and vision models. This success has inspired the application of transformers to time-series forecasting, with significant contributions from [28, 25, 29, 14, 19, 15, 27]. Informer [16] introduces ProbSparse attention, reducing complexity for long sequence forecasting, while [29] leverages Fourier domain sparsity. PatchTST [19] uses patch-based tokens for interpretable time-series tokenization, while iTransformer [16] emphasizes inter-channel relationships by treating each variate as a single token. In zero-shot forecasting, studies like [21, 20, 9, 5, 2] have advanced the field. Chronos [2] utilizes novel tokenization, quantization, and synthetic data generation to adapt LLMs for univariate forecasting, while ForecastPFN [5] relies entirely on synthetic datasets, building on Prior-data Fitted Networks (PFNs) [18]. Our work, introducing the first multivariate Prior-data Fitted Network, builds on these advances to deliver strong zero-shot and few-shot performance in MTS forecasting.

3 Proposed Method

This work relies on two key aspects: a multivariate synthetic time series data generation mechanism that encapsulates inter- and intra-channel dependencies common across real time series data, and an architecture capable of generalization to real datasets when trained on such a dataset. Our methodology relies heavily on prior-data-fitted networks (PFNs). However, due to space constraints, the PFN formalization and background are provided in Appendix A. The overview of architecture and algorithm are in Figure 1. For the ablations, please refer to Appendix D.3 and G.

3.1 Synthetic MTS Data Generation

In synthetic multivariate time series (MTS) data generation, we aim to create variates that are both realistic (with periodic patterns and trends) and correlated, reflecting real-world MTS characteristics. The first goal is addressed by KernelSynth, as used in Chronos [2], which generates diverse univariate synthetic time-series data by composing kernels with binary operators (e.g., addition, multiplication). Unlike the kernel composition method used for structure discovery in nonparametric regression [6], KernelSynth generates realizations from these kernels. For instance, a linear kernel combined with a periodic kernel results in a time series with both trends and seasonality, while multiplying a squared-exponential kernel with a periodic one creates locally periodic patterns [6]. Chronos assembles various kernels (Linear, Periodic, Squared-Exponential, etc.) with different parameters (e.g., daily, weekly periodicities) to define Gaussian processes.

To address the second goal—generating correlated variates—we use a generative Gaussian model called the linear model of coregionalization (LMC) [10]. LMC produces outputs as linear combinations of independent latent random functions: $C_i(\boldsymbol{t}) = \sum_{j=1}^{L} \alpha_{i,j} l_j(\boldsymbol{t})$, resulting in a positive semi-definite covariance matrix [1]. In our algorithm, LMC-Synth (see Algorithm 1), we restrict to convex combinations to avoid scaling issues. Latent functions are generated using KernelSynth, allowing for cases of small or nonexistent correlations, with independent variates modeled as $L = N$ and $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$. This flexibility is essential for MTS problems with varying degrees of correlation. LMC-Synth samples the number of latent functions from a Weibull distribution and α values from a Dirichlet distribution, with bounds to avoid skewed results. A more detailed take of data generation is provided in Appendix B.

3.2 Architecture for TimePFN

Our goal was to design an architecture that effectively extracts time-series features for MTS forecasting while generalizing to real-world datasets. The PFN framework, leveraging LMC-Synth for large-scale MTS data synthesis, overcomes data scarcity limitations. Unlike prior MTS models constrained by limited data and forced to simplify to avoid overfitting, we can now expand the architecture and incorporate components that improve forecasting on new datasets. ***TimePFN*** shares similarities with PatchTST [19], but it diverges in two key aspects: convolutional filtering before patching and channel-mixing. A more detailed take of the architecture is provided in Appendix C.

Convolutional Filtering. Given an MTS dataset $X = [x_1 \dots x_N]$ in $\mathbb{R}^{L \times N}$, where L is the sequence length and N the number of variates, we apply 1D convolutions with shared weights across all variates. This is followed by magnitude max pooling. In ***TimePFN***, each $x_i \in \mathbb{R}^L$ becomes $\bar{x}_i \in \mathbb{R}^{(C+1) \times L}$, with C rows from convolutions and one row retaining the original x_i, similar to skip connections in NLP [7]. We used $C = 9$. This approach helps capture common time-series features across datasets, improving generalization.

Table 1: MTS forecasting results of TimePFN and comparable architectures with best results in bold. Input and prediction lengths are set to be 96. *TimePFN* demonstrates remarkable performance in budget-limited settings, as well as with the full dataset, particularly in scenarios involving a large number of variates.

	Dataset	ECL		Weather		Traffic		Solar-Ener.		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
z.s.	TimePFN	**0.315**	**0.383**	**0.209**	**0.255**	**1.108**	**0.613**	0.941	**0.730**	**0.453**	**0.439**	**0.328**	**0.362**	**0.637**	**0.512**	**0.212**	**0.291**
	Naive	1.587	0.945	0.259	0.254	2.714	1.077	1.539	0.815	1.294	0.713	0.431	0.421	1.213	0.664	0.266	0.327
	SeasonlN	1.618	0.964	0.268	0.263	2.774	1.097	1.599	0.844	1.325	0.727	0.445	0.431	1.227	0.673	0.274	0.334
	Mean	0.845	0.761	0.215	0.271	1.410	0.804	**0.910**	0.734	0.700	0.558	0.352	0.387	0.693	0.547	0.229	0.307
Budget =50	TimePFN	**0.235**	**0.322**	**0.190**	**0.235**	**0.746**	**0.468**	**0.429**	**0.450**	**0.438**	**0.429**	**0.324**	**0.359**	**0.419**	**0.418**	**0.195**	**0.276**
	iTransf.	0.278	0.360	0.237	0.278	0.801	0.499	0.513	0.479	0.838	0.617	0.410	0.422	0.884	0.608	0.268	0.337
	PatchTST	0.667	0.646	0.221	0.269	1.295	0.746	0.810	0.669	0.778	0.587	0.372	0.401	0.656	0.528	0.231	0.310
	DLinear	0.406	0.463	0.742	0.612	1.888	0.937	0.956	0.813	1.404	0.881	3.928	1.383	1.332	0.846	3.484	1.290
	FEDForm	0.908	0.758	0.306	0.381	1.587	0.874	0.972	0.757	0.676	0.570	0.424	0.468	0.745	0.589	0.291	0.387
	Informer	1.226	0.896	0.464	0.511	1.714	0.901	0.887	0.783	1.172	0.819	2.045	1.093	1.003	0.745	1.590	0.995
	Autoform	0.729	0.675	0.322	0.401	1.600	0.883	1.065	0.808	0.607	0.560	0.492	0.506	0.763	0.592	0.316	0.407
Budget =500	TimePFN	**0.190**	**0.283**	**0.178**	**0.222**	**0.487**	**0.335**	**0.269**	**0.305**	**0.401**	**0.412**	**0.311**	**0.352**	**0.360**	**0.386**	**0.185**	**0.268**
	iTransf.	0.200	0.284	0.211	0.248	0.514	0.354	0.307	0.334	0.489	0.470	0.361	0.394	0.569	0.494	0.231	0.310
	PatchTST	0.236	0.320	0.210	0.246	0.740	0.455	0.321	0.353	0.596	0.515	0.358	0.392	0.369	0.386	0.190	0.275
	DLinear	0.235	0.328	0.335	0.394	1.312	0.727	0.622	0.656	0.749	0.609	1.098	0.712	0.817	0.621	0.870	0.626
	FEDForm	0.317	0.407	0.265	0.341	0.888	0.548	0.821	0.706	0.444	0.452	0.358	0.401	0.674	0.542	0.238	0.322
	Informer	0.869	0.760	0.320	0.393	1.411	0.774	0.318	0.385	0.913	0.713	1.311	0.940	0.704	0.595	1.121	0.803
	Autoform	0.303	0.396	0.237	0.312	0.896	0.549	0.950	0.787	0.456	0.456	0.339	0.384	0.672	0.534	0.223	0.308
Budget = All	TimePFN	**0.138**	**0.137**	**0.166**	**0.208**	**0.392**	**0.260**	0.203	0.219	0.402	0.417	**0.293**	**0.343**	0.392	0.402	0.180	0.262
	iTransf.	0.147	0.239	0.175	0.215	0.393	0.268	0.201	0.233	**0.387**	0.405	0.300	0.349	0.342	0.376	0.185	0.272
	PatchTST	0.185	0.267	0.177	0.218	0.517	0.334	0.222	0.267	0.392	**0.404**	**0.293**	**0.343**	**0.318**	**0.357**	**0.177**	**0.260**
	DLinear	0.195	0.278	0.341	0.412	0.690	0.432	0.286	0.375	0.400	0.412	0.357	0.406	0.344	0.371	0.195	0.293
	FEDForm	0.196	0.310	0.227	0.313	0.573	0.357	0.242	0.342	0.380	0.417	0.340	0.386	0.363	0.408	0.191	0.286
	Informer	0.327	0.413	0.455	0.481	0.735	0.409	**0.190**	**0.216**	0.930	0.763	2.928	1.349	0.623	0.559	0.396	0.474
	Autoform	0.214	0.327	0.273	0.344	0.605	0.376	0.455	0.480	0.440	0.446	0.364	0.408	0.520	0.490	0.233	0.311
	#ofVariates	321		21		862		137		7		7		7		7	

Patch Embeddings. From $\bar{x}_i \in \mathbb{R}^{(C+1)\times L}$, overlapping patches of size P and stride S are extracted as in [19]. Each patch, sized $\mathbb{R}^{(C+1)\times P}$, is flattened and passed through a 2-layer feedforward network, with positional encodings [24] added to embed channel-wise and temporal information. We used $(P = 16, S = 8)$.

Channel-mixing. Unlike PatchTST [19], where channels are processed independently, we feed all tokens into the transformer encoder, allowing inter-variate attention.

Transformer Encoder. A multihead transformer encoder with layer normalization [3] and skip connections [7] is used for stability. After encoding, tokens are rearranged by channel and passed through a two-layer feedforward network with shared weights across variates.

Architectural Details. We normalize each variate x_i to zero mean and unit variance, following [11], to address distribution shifts between synthetic and test datasets [16, 19]. De-normalization is applied before forecasting. *TimePFN* has fixed input sequence and forecasting lengths but can handle arbitrary numbers of variates. Although trained with a fixed channel size ($N = 160$), *TimePFN* can forecast with both fewer and more channels. For test cases with more channels, data is split into segments of size at most N, processed separately, and stacked afterwards.

4 Experiments

In MTS evaluations, we focused on forecasting 96 future time steps using a 96-step MTS input. We trained a single *TimePFN* model on a large synthetic dataset generated by LMC-Synth, consisting of 15,000 datasets (length 1024, 160 channels), being augmented with independent variates ($\approx 25\%$). Using a sliding window of 192 (96 input, 96 output), we extracted 1.5 million data points, training the model with MSE loss. In few-shot evaluations, we fine-tuned *TimePFN* with a specified data budget, using the same hyperparameters for all settings. The details of experiments (see Appendix D) and ablations (see Appendix D.3 and G) are provided in the appendix.

Baselines. We tested *TimePFN* on nine widely-used MTS forecasting datasets: ETTh1, ETTh2, ETTm1, ETTm2, Weather, Solar Energy [13], ECL, and Traffic [25] (details in the appendix). With no MTS PFN available, we compared *TimePFN* with state-of-the-art models: FEDformer [29], Autoformer [25], Informer [28], PatchTST [19], iTransformer [16], and DLinear [26]. We evaluated them across different data budgets (50, 100, 500, 1000 points). For zero-shot evaluations, we included Naive (repeating the last element), and Seasonal Naive (repeat with seasonality 7) baselines

Table 2: Zero-shot results of TimePFN on univariate time-series forecasting with input length = 36. TimePFN-96 has input length of 96. The errors are averaged over forecasting lengths of $\{6, 8, 14, 18, 24, 36, 48\}$. We do not report the MAE for Meta-N-Beats, as these results, sourced from [5], did not include that metric. The best results are in bold (excluding TimePFN-96). Weather results are multiplied by $\times 10^2$.

Dataset	TimePFN-36		TimePFN-96		ForecastPFN		Chronos-smll.		Meta-N-Beats	SeasonalNaive		Naive	
Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MSE	MAE	MSE	MAE
ECL	**0.752**	**0.703**	0.509	0.549	1.416	0.958	1.152	0.792	0.909	1.559	0.995	1.211	0.829
Weather	0.042	1.381	0.046	1.477	0.084	1.999	0.036	1.136	1.400	0.045	1.352	**0.035**	**1.123**
Traffic	**1.503**	**1.032**	0.414	0.503	4.521	1.742	3.103	1.364	2.913	4.301	1.771	3.330	1.463
ETTh1	**0.029**	0.130	0.030	0.133	0.102	0.237	0.061	0.155	0.177	0.039	0.151	0.031	**0.128**
ETTh2	**0.126**	**0.273**	0.086	0.224	0.434	0.517	0.207	0.321	0.480	0.279	0.408	0.215	0.336

[5, 2]. Although trained for multivariate forecasting, we tested *TimePFN* on univariate tasks (C=1), comparing it with ForecastPFN [5] and Chronos [2]. We padded *TimePFN* 's input with the mean of 36-length MTS to match ForecastPFN's 36-length sequence, and evaluated over forecast lengths of 6 to 48 time-steps, reporting averaged MSE and MAE values (details in the appendix). We also ran tests with a non-padded 96-step sequence. Lastly, we used Meta-N-Beats [20] results from [5] for baseline comparison, while evaluating all other results independently.

4.1 Main Results

In MTS forecasting, we compared *TimePFN* with various baselines in zero-shot settings, different data budgets, and the full dataset. Table 1 shows results for zero-shot, data budgets of 50 and 500, and full dataset usage. Extended results, including budgets of 100 and 1000, are in the Appendix under *Extended Results*. With a data budget of 50 and 500, *TimePFN* outperforms all transformer-based models and DLinear. Using the full dataset, *TimePFN* achieves the best results in four datasets, outperforming PatchTST's performance. Given the fixed hyperparameters for *TimePFN* across all datasets and our use of the best baseline results among our runs and the reportings of [16], *TimePFN* 's performance is notable. *TimePFN* excels in datasets with more variates, while PatchTST performs best in ETT datasets, as expected due to their respective designs for channel mixing and independence. PatchTST's weaker performance on Traffic also supports this hypothesis.

In zero-shot settings, *TimePFN* outperforms all baselines except in the Solar-Energy. Solar-Energy's sudden spikes, due to factors like sunrise and sunset, are not well predicted by *TimePFN* trained on LMC-Synth data. However, these patterns are within the scope of changepoint kernels in Gaussian processes, suggesting future improvements.

4.2 Univariate Time-Series Forecasting

Though *TimePFN* was trained for MTS forecasting on a synthetic dataset with a channel size of 160, we tested it in a zero-shot scenario for univariate time-series forecasting ($C = 1$). We used a sequence length of 36, as in ForecastPFN [5], padding the remaining 60 sequence lengths with the mean of the input to prevent scaling issues. This model is *TimePFN*-36. We also tested *TimePFN* without padding, using a sequence length of 96 (*TimePFN*-96), as shown in Table 2. Table 2 shows *TimePFN* surpassing models trained specifically for univariate time series forecasting, demonstrating its robust generalization and zero-shot performance. Further evaluations, including error breakdowns for different sequence lengths, are in the Appendix E under *Extended Results*.

5 Conclusion

In this work, we show that with large-scale synthetic training and a suitable architecture for extracting time-series features, fine-tuning with as few as 50–500 examples can achieve competitive multivariate time series forecasting performance. We introduce LMC-Synth, a novel method for generating large-scale synthetic MTS data with realistic intra- and inter-channel dependencies, using Gaussian processes and a linear coregionalization model. Our architecture, *TimePFN*, uses 1D convolutions and channel-mixed patching, exhibiting strong zero-shot performance in both MTS and univariate forecasting. To our knowledge, *TimePFN* is the first multivariate time-series PFN. As a future work, we will explore diversifying synthetic data generation to model sudden changes and multi-scale issues, and expand our work into MTS foundation models.

Acknowledgements

This work was supported in part by the NSF grants CCF-2046816, CCF-2403075, the Office of Naval Research grant N000142412289, and gifts by Open Philanthropy and Google Research.

References

[1] M.A. Álvarez, L. Rosasco, and N.D. Lawrence. *Kernels for Vector-Valued Functions: A Review*. Foundations and Trends® in Machine Learning Series. Now Publishers, 2012.

[2] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Syndar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[5] Samuel Dooley, Gurnoor Singh Khurana, Chirag Mohapatra, Siddartha V Naidu, and Colin White. Forecastpfn: Synthetically-trained zero-shot forecasting. In *Advances in Neural Information Processing Systems*, 2023.

[6] David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1166–1174, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[8] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.

[9] Xiaoyong Jin, Youngsuk Park, Danielle Maddix, Hao Wang, and Yuyang Wang. Domain adaptation for time series forecasting via attention sharing. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 10280–10297. PMLR, 17–23 Jul 2022.

[10] A.G. Journel and C.J. Huijbregts. *Mining Geostatistics*. Blackburn Press, 2003.

[11] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[13] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, page 95–104, New York, NY, USA, 2018. Association for Computing Machinery.

[14] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[15] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2022.

[16] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[18] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022.

[19] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

[20] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. Meta-learning framework with applications to zero-shot time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9242–9250, May 2021.

[21] Bernardo Pérez Orozco and Stephen J. Roberts. Zero-shot and few-shot time series forecasting with ordinal regression recurrent neural networks. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pages 503–508, 2020.

[22] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[24] Zelun Wang and Jyh-Charn Liu. Translating math formula images to latex sequences using deep neural networks with sequence-level training, 2019.

[25] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with Auto-Correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, 2021.

[26] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023.

[27] Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *International Conference on Learning Representations*, 2023.

[28] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, volume 35, pages 11106–11115. AAAI Press, 2021.

[29] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *Proc. 39th International Conference on Machine Learning (ICML 2022)*, 2022.

A Prior-data Fitted Networks for MTS Forecasting

Let $\mathcal{D} := \{t, X_t\}_{t=1}^{T}$ represent an N-channel multivariate time series data spanning a time horizon T, where $X_t := [x_{t,1}, \ldots, x_{t,N}]$. Each $x_{t,i}$ is potentially causally dependent on previous time steps and on one another. Given the data $\{t, X_t\}_{t=1}^{\widetilde{T}}$ where $\widetilde{T} < T$, the task is to forecast $X_{\widetilde{T}+1}, \ldots, X_T$. We tackle this problem using a Bayesian framework. Assuming a hypothesis space Ω with a prior distribution $p(\omega)$, each hypothesis $\omega \in \Omega$ models a multivariate time series (MTS) generating process, i.e., $X_t = \omega(t)$. For example, Ω could represent the space of hypotheses for vector autoregression (VAR) models, where a particular instance $\omega \in \Omega$ corresponds to a specific VAR process, such as VAR(2), and data $\mathcal{D}$ can be generated via this process. Now, given a data $\mathcal{D} := \{t, X_t\}_{t=1}^{\widetilde{T}}$ where $\widetilde{T} < T$, the posterior predictive distribution (PPD) of $\boldsymbol{x} \in \mathbb{R}^N$ at time T is $p(\cdot \mid T, \mathcal{D})$. By Bayes' theorem,

$$p(\boldsymbol{x} \mid T, \mathcal{D}) \propto \int_{\Omega} p(\boldsymbol{x} \mid T, \omega) p(D \mid \omega) p(\omega) d\omega \tag{1}$$

As shown by [18, 8, 5], the posterior predictive distribution (PPD) is approximated using prior fitting networks (PFNs) as follows: We iteratively sample a hypothesis ω from the hypothesis space Ω according to the probability $p(\omega)$. Next, we generate a prior dataset $\mathcal{D}$ from this hypothesis, denoted as $\mathcal{D} \sim p(\mathcal{D} \mid \omega)$. We then optimize the parameters of the PFN on these generated datasets using standard methods. The time series dataset is divided into input and output parts, where $\mathcal{D}_{input} := \{t, X_t\}_{t=1}^{\widetilde{T}}$ and $\mathcal{D}_{output} := \{t, X_t\}_{t=\widetilde{T}+1}^{T}$. Subsequently, we train the PFN to forecast $\mathcal{D}_{output}$ from $\mathcal{D}_{input}$ using standard time-series transformer training techniques, aiming to minimize the mean-squared error loss as our optimization objective, following the setting of [5].

In our work, we define the hypothesis space Ω as consisting of single-input, multi-output Gaussian processes represented by the linear model of coregionalization (LMC) [10]. Our choice is driven by the representational power of Gaussian processes and their ability to generate a diverse range of time series through the LMC framework.

B Synthetic MTS Data Generation

In synthetic MTS (multivariate time series) data generation, our goal is twofold. First, we strive to create variates that are realistic, exhibiting periodic patterns, trends, and other common features found in real-world data. Second, we aim for these variates to be correlated with one another, which better represents MTS data characteristics. Fortunately, our first goal is addressed by a method called KernelSynth. Chronos [2] uses KernelSynth to enrich its training corpus by randomly composing kernels using binary operators (such as addition and multiplication) to generate diverse, univariate synthetic time-series data. This method is essentially the inverse of the kernel composition approach described in [6], where kernel compositions are used for structure discovery in nonparametric regression. In contrast, KernelSynth focuses on generating realizations from these kernels. For example, combining a linear kernel with a periodic kernel results in a pattern that exhibits both a linear trend and sinusoidal seasonality. Similarly, multiplying a squared-exponential kernel with a periodic kernel creates locally periodic patterns [6]. Chronos aggregates kernels of various types—such as Linear, Periodic, Squared-Exponential, Rational, and Quadratic—and with different parameters (such as daily, weekly, and monthly periodic kernels) in a kernel bank, composing them as described above to define the Gaussian processes.

However, generating a MTS time-series data is yet to be addressed. To address the second goal, generating variates that are correlated in a realistic manner, we use a generative Gaussian modelling, called linear model of coregionalization (LMC), which is developed initially in the field of geostatistics [10]. For ease of understanding, we adopt the time-series notation we used above. In LMC, the outputs are obtained as linear combinations of independent latent random functions. In other words, given $\boldsymbol{t} \in \mathbb{R}^T$, the outputs in each channel $\{C_i(\boldsymbol{t})\}_{i=1}^{N}$ is the linear combination of L latent functions

$$C_i(\boldsymbol{t}) = \sum_{j=1}^{L} \alpha_{i,j} l_j(\boldsymbol{t}). \tag{2}$$

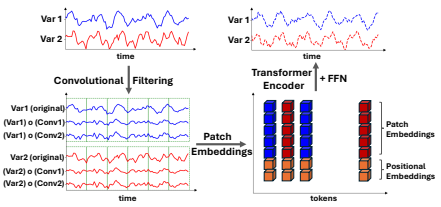


Figure 1: Illustration of the architecture of TimePFN. Variates are filtered with 1D convolutions, to be patched with overlapping strides, following [19]. They are then fed into transformer encoder with channel mixing, with the final forecast coming from the final feedforward network.

Algorithm 1 LMC-Synth

Input: Number of variates N, time-series length T, Weibull shape parameter β, Weibull scale parameter λ, (min, max) value of dirichlet concentration parameter (d_{min}, d_{max}), minimum number of latent functions m, maximum number of kernel composition in KernelSynth J

Output: Synthetic MTS $\boldsymbol{C}$ with N variates and length T

$L \sim \max(\min(\text{Weibull}(\beta, \alpha), N), m)$
$d \sim U(d_{min}, d_{max})$
for j $\in \{1 \dots L\}$ **do**
  $l_j(\boldsymbol{t}) \leftarrow \text{KernelSynth}(J, T)$
end for
for i $\in \{1 \dots N\}$ **do**
  $[\alpha_{i,1} \dots \alpha_{i,L}] \sim \text{Dir}(d)$
  $C_i(\boldsymbol{t}) \leftarrow \sum_{j=1}^{L} \alpha_{i,j} l_j(\boldsymbol{t})$
end for
return $\{C_i(\boldsymbol{t})\}_{i=1}^{N}$

Observe that, since latent functions are independent with zero-mean the resulting output covariance is a well-defined PSD function with zero-mean [1]. In our synthetic data generation algorithm, to avoid scaling issues, we restrict ourselves to convex combinations. Thus, for each i, we have $\alpha_{i,1} + \dots \alpha_{i,L} = 1$ with $\alpha_{i,j} \geq 0$, meaning that the outputs lie in the convex hull of latent functions l_j's. We generate the latent functions based on KernelSynth's algorithm due to it's extensive descriptive value. Note that the LMC formulation encapsulates the cases where the correlations between different variates are small or nonexistent. Specifically, the case where each variate is independent from the rest corresponds to $L = N$ with $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$. Such a modelling is important, as some MTS data have strong correlation between different variates, whereas others have small or non-existent correlation.

In our algorithm, LMC-Synth, we sample the number of latent functions from a Weibull distribution and $[\alpha_{i,1} \dots \alpha_{i,L}]$ from a Dirichlet distribution. To avoid highly skewed cases, we impose upper and lower bounds on the possible number of latent functions. Since the uncorrelated setting of $L = N$ with $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$ is crucial for modeling MTS problems with low correlation among variates, we also generate data under this setting. Incorporating this extra setting is shown to yield the strongest performance in zero-shot settings.

C Architecture

In designing the architecture, our main principle was to create a system capable of extracting time-series features useful for MTS forecasting. Through this, we aimed for the architecture to achieve better generalization when applied to real-world datasets. The primary advantage of the PFN framework in our case is that, since synthesizing large-scale synthetic MTS data is feasible with LMC-Synth, we are no longer constrained by data scarcity. Previous MTS models were compelled to balance model complexity with their datasets due to limited data, often restricting the use of certain components or their quantity (such as the number of transformer layers) to avoid overfitting. However, with access to large-scale MTS data, we can expand our architecture and freely incorporate additional components that we believe will improve forecasting performance on new datasets. In light of this, we proceed to explain our architecture and design choices.

The *TimePFN* model resembles PatchTST [19] in several aspects when processing MTS data, but it differs significantly in two areas: our convolutional filtering of the variates prior to patching and channel-mixing.

Convolutional Filtering. Before patching, consider an MTS dataset $X = [x_1 \dots x_N]$ in $\mathbb{R}^{L \times N}$, where L is the length and N is the number of variates. We apply learnable 1D convolution operations to each variate, with convolutional weights shared across all variates. After convolutions, we apply 1D magnitude max pooling to each newly generated variate, followed by a new set of 1D convolutions. In *TimePFN*, each $x_i \in \mathbb{R}^L$ is transformed into $\bar{x}_i \in \mathbb{R}^{(C+1) \times L}$, where C rows come from 1D

convolutional operations and magnitude max pooling, whereas one row is the original x_i. We keep the original x_i to not loose any information, analogous to skip connections used in NLP [7]. In practice, we used $C = 9$. Filtering with convolutions is a valuable tool in time-series analysis. Many operations, such as differencing to de-trend data, can be effectively represented by convolutions. We utilized this approach to extract common time-series features across various datasets, thereby improving the generalization capability of our model.

Patch Embeddings. Given $\bar{x}_i \in \mathbb{R}^{(C+1)\times L}$, we extract overlapping patches of size P with a stride of S, following the settings described in [19]. Each patch thus has dimensions $\mathbb{R}^{(C+1)\times P}$, and a total of $\left\lfloor \frac{L-P}{S} + 2 \right\rfloor$ patches are extracted from a single variate. In total, we get $N \times \left\lfloor \frac{L-P}{S} + 2 \right\rfloor$ patches. Each patch is then flattened and fed into a 2-layer feedforward neural network to be mapped to embedding dimension D. We add 2D sinusoidal positional encodings [24] to the embeddings to correctly capture channel-wise and temporal information. In practice, we used $(P = 16, S = 8)$, similar to [19].

Channel-mixing. Unlike PatchTST [19], where the tokens from each channel are fed independently into a transformer encoder, we input all tokens into the transformer encoder after applying the positional encodings described above. Consequently, tokens from different variates can attend to each other.

Transformer Encoder. We employ a naive multihead transformer encoder, incorporating layer normalizations [3] and skip connections [7] to improve training stability. After feeding the tokens into the multilayer encoder, we rearrange them into their respective channels and apply a channel-wise flattening operation. This is followed by a two-layer feedforward network that processes the flattened variate representations using shared weights (single FFN is applied to all variates).

Normalization. We normalize each variate x_i to have zero mean and unit standard deviation prior to any other process described above, as recommended by [11], to alleviate the impact of distribution shifts between our synthetic dataset and test examples [16, 19]. Before forecasting, we revert the time series to its original scale by de-normalizing.

Architectural Details. Due to our architectural specifications, *TimePFN* has fixed input sequence and forecasting lengths. However, it can accept an arbitrary number of variates. Thus, although we trained *TimePFN* with a synthetic dataset of a fixed channel size ($C = 160$), it can forecast with both fewer and more channels than those used in its training data. When forecasting with a number of channels $\bar{C} \leq C$, we directly input the data to *TimePFN*. To mitigate the effects of distribution shifts when forecasting with more channels at test time, we process the data by splitting it into non-overlapping channels of size at most C. If the test data has $\bar{C}$ channels, we divide it into $\left\lfloor \frac{\bar{C}}{C} \right\rfloor + 1$ segments, input them separately, and then stack them afterwards.

D Experiments

In all MTS evaluations, our primary objective is to forecast a horizon of 96 time steps using an MTS input of 96 time steps. We trained a single *TimePFN* model on a large-scale, multivariate synthetic dataset generated by LMC-Synth and conducted all experiments using this model. We generated 15,000 synthetic datasets with a length of 1024 and a channel size of 160 from LMC-Synth, further augmenting with datasets having independent variates as in the case $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$. The independent data comprises approximately 25% of the purely correlated data. During training, we extracted time-series input and output pairs using a sliding window of size 192 (96 for input, 96 for output), resulting in approximately 1.5 million synthetic data points. We trained the model to forecast the MTS output based on the given input using MSE loss with our 160 channel synthetic dataset. Training a single *TimePFN* of 8 transformer layers takes around 10 hours on L40S GPU.

In the few-shot evaluations, we fine-tuned *TimePFN* using the specified data budget. We did not perform any hyperparameter tuning on *TimePFN*, and the same set of hyperparameters was used in all few-shot settings. Details about the model hyperparameters are provided in the appendix.

Benchmark Datasets. We evaluated *TimePFN* on nine widely-used, real-world benchmark datasets for MTS forecasting. These datasets include ETTh1, ETTh2, ETTm1, ETTm2 (collectively referred to as ETT, representing Electricity Transformer Temperature), Weather, Solar Energy, ECL (Electricity Consuming Load), Exchange, and Traffic. The Solar Energy dataset was introduced by [13], while

the others were introduced by [25]. We provide the specifications of these datasets in the appendix section *Datasets*.

Baselines. Since no MTS PFN is available, we compared *TimePFN* with state-of-the-art transformer-based MTS forecasting models, including FEDformer [29], Autoformer [25], Informer [28], PatchTST [19], and iTransformer [16]. We evaluated these models across the entire dataset and at various data budgets, including 50, 100, 500, and 1000 data points. For instance, at a data budget of 500, the model is trained using 500 MTS input and output pairs. Additionally, we included DLinear [26], a linear model, as part of our baseline. Given its lower complexity, we consider it a strong baseline for smaller data budgets.

For smaller data budgets and our zero-shot evaluations, we incorporated three algorithmic baselines as suggested by [5] and [2]: Mean, Naive, and Seasonal Naive. These baselines are applied independently to each variate. The Mean baseline forecasts by repeating the mean value of the input variate. The Naive approach forecasts by repeating the last value of the input variate. In the Seasonal Naive method, we assume a periodicity of seven.

Although *TimePFN* is specifically trained for multivariate time-series forecasting, we also evaluated its performance on univariate forecasting (C=1) to demonstrate its robust generalization capabilities. In this context, we compared it with ForecastPFN [5] and Chronos [2], two state-of-the-art univariate zero-shot forecasters. ForecastPFN utilizes an input sequence length of 36, while *TimePFN* operates with a sequence length of 96. To accommodate this discrepancy, we padded the additional 60 time steps with the mean value of the input sequence when running *TimePFN*. These models are evaluated over forecast lengths of 6, 8, 14, 18, 24, 36, and 48. We used the smaller version of Chronos. The full results are detailed in the appendix, while in the main text, we report the averaged MSE and MAE values across these forecast lengths. Furthermore, to showcase the complete forecasting performance of *TimePFN*, we also conducted runs with a non-padded sequence length of 96.

Additionally, we used the forecasting results of Meta-N-Beats [20], from the reporting of [5] as baseline comparisons, while we evaluated all other results ourselves independently.

Experimental Setting. When comparing *TimePFN* with the aforementioned baselines, we use the hyperparameters reported in their official codebases. We re-run the experiments with limited budgets and by utilizing the entire training dataset. [16] presents the forecasting results for the mentioned transformer-based MTS architectures using the full training dataset. We re-run all the experiments and selected the best results from both our run and their report to ensure that the performance of other architectures is not underreported when we use the entire training set. Our unaltered results are included in the appendix.

In *TimePFN*, we use a single model with fixed hyperparameters that is trained only once on our large-scale multivariate synthetic dataset. In few-shot evaluations, we fine-tune *TimePFN* with a given data budget, maintaining the same hyperparameters across different datasets. In all evaluations except for univariate cases, we report the forecasting errors for the next 96 time steps, given a multivariate time series (MTS) of sequence length 96. Our implementation details and further experimental settings such as hyperparemeters are reported in the Appendix: *Implementation Details*.

D.1 Main Results

In MTS forecasting, we compared *TimePFN* with various baselines in zero-shot settings, as well as with different data budgets, and by utilizing the entire dataset. Table 1 presents our results for zero-shot settings, data budgets of 50 and 500, and scenarios using the entire dataset. Our comprehensive results, which also include data budgets of 100 and 1000, can be found in the Appendix under the section *Extended Results*. With a data budget of 50, *TimePFN* outperforms all transformer-based architectures and DLinear. With a data budget of 500, it surpasses all baselines. When utilizing the entire dataset, *TimePFN* achieves the best results in four datasets, equaling the performance of PatchTST. Given that we fine-tuned *TimePFN* with fixed hyperparameters across all datasets, and selected the best results from the baselines and the findings reported in [16], the performance of *TimePFN* is noteworthy. We observe that *TimePFN* excels in datasets with a greater number of variates and a more multivariate nature, while PatchTST primarily excels in ETT datasets. This outcome is anticipated, as *TimePFN* is designed to incorporate channel mixing, whereas PatchTST is designed with channel independence. Indeed, the lower forecasting performance of PatchTST on Traffic dataset supports this hypothesis.

	Dataset	ECL		Weather		Traffic		Solar-Ener		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
z.s.	T.PFN	**0.315**	**0.383**	**0.209**	**0.255**	**1.108**	**0.613**	**0.941**	**0.730**	**0.453**	**0.439**	0.328	**0.362**	**0.637**	**0.512**	**0.212**	**0.291**
z.s.	T.PFNw/oC	0.653	0.637	0.221	0.271	1.287	0.757	1.197	0.829	0.608	0.517	0.338	0.374	0.771	0.565	0.224	0.307
z.s.	P.TST-PFN	0.470	0.522	0.212	0.262	1.172	0.702	1.014	0.787	0.554	0.501	**0.322**	0.366	0.746	0.560	0.215	0.301

Table 3: In T.PFNw/oC (TimePFN-w/o-Convolution), we eliminate the convolutional operator that is normally applied to the initial input variates. In P.TST-PFN (PatchTST-PFN), we train a PatchTST model to evaluate the significance of channel-mixing and the appropriateness of our architecture for PFNs. Both the sequence length and the forecasting length are set to 96. T.PFN stands for TimePFN.

	Dataset	ECL		Weather		Traffic		Solar-Ener		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
z.s.	T.PFN	**0.315**	**0.383**	**0.209**	**0.255**	**1.108**	**0.613**	**0.941**	**0.730**	**0.453**	**0.439**	0.328	**0.362**	**0.637**	**0.512**	**0.212**	**0.291**
z.s.	T.PFN-I.	0.350	0.416	0.214	0.260	1.180	0.651	1.197	0.829	0.468	0.447	**0.326**	0.363	0.761	0.542	0.215	0.295

Table 4: T.PFN-I. (TimePFN-ind) is the model trained using only independent variates, while the other model is our standard one, which we used throughout the experiments. Both models have a sequence and forecasting length of 96. T.PFN stands for TimePFN.

In zero-shot settings, *TimePFN* outperforms all zero-shot baselines except on the Solar-Energy dataset, with Solar-Energy being in close proximity. We observed that the Solar-Energy data exhibits sudden spikes e.g. as a function of sun rising or going down. Our model, based on its training data from the LMC-Synth prior, fails to anticipate such sudden spikes. However, these spiky behavior is well within the capabilities of changepoint kernels in Gaussian processes, suggesting a clear path for future improvements.

D.2 Univariate Time-Series Forecasting

Although *TimePFN* was specifically trained for MTS forecasting using a synthetic dataset with a channel size of 160, we also tested it in a zero-shot scenario for univariate time-series forecasting where $C = 1$. Moreover, we used the sequence length of 36 that ForecastPFN [5] was specifically trained on. To accommodate this sequence length, we padded the remaining $96 - 36 = 60$ sequence lengths with the mean value of the input time-series to mitigate any scaling issues, and named this model configuration *TimePFN*-36. To demonstrate the full performance of our model, we included results for *TimePFN* without padding using a sequence length of 96, referred to as *TimePFN*-96 in Table 2. All other results were reported with a sequence length of 36.

As demonstrated in Table 2, *TimePFN* outperforms models that were specifically trained for univariate time series forecasting, which attests to its robust generalization and zero-shot performance. Our extensive evaluations, which detail the errors for different sequence lengths, can be found in the appendix under the section *Extended Results*.

D.3 Ablation Study

Training a single *TimePFN* model requires approximately 10 hours on a single L40S GPU, which limited our capacity for ablation studies. Nevertheless, we conducted two types of key ablations: the first type focused on the architecture, while the second type focused on the synthetic data generation.

Architectural Ablation. In the first part, we first aim to understand the impact of our 1D convolutional operation applied to time-series variates before any patching. To do this, we remove the operation and train a *TimePFN*-convolutionless model, then report the zero-shot results in Table 3. We observe that without the convolutional operation, the zero-shot performance significantly decreases. Additionally, since our architecture differs from that of [19] particularly in terms of channel mixing, we trained the PatchTST architecture to assess the impact of channel mixing on zero-shot forecasting performance. As seen in Table 3, both of our ablation experiments supports our model design principles and underscores the usefulness of *TimePFN*'s architecture for synthetic learning.

Synthetic Dataset Ablation. To understand whether the synthetic data generation algorithm LMC-Synth gives any benefits over just using the variates generated by KernelSynth [2] independently in each channel, we trained *TimePFN* with using data where each channel is generated indepen-

	Dataset	ECL		Weather		Traffic		Solar		Exchange		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
z.s.	TimePFN	**0.315**	**0.383**	**0.209**	**0.255**	**1.108**	**0.613**	0.941	**0.730**	0.105	0.229	**0.453**	**0.439**	**0.328**	**0.362**	**0.637**	**0.512**	**0.212**	**0.291**
	Naive	1.587	0.945	0.259	0.254	2.714	1.077	1.539	0.815	**0.081**	**0.196**	1.294	0.713	0.431	0.421	1.213	0.664	0.266	0.327
	SeasonalN.	1.618	0.964	0.268	0.263	2.774	1.097	1.599	0.844	0.086	0.204	1.325	0.727	0.445	0.431	1.227	0.673	0.274	0.334
	Mean	0.845	0.761	0.215	0.271	1.410	0.804	**0.910**	0.734	0.139	0.269	0.700	0.558	0.352	0.387	0.693	0.547	0.229	0.307
Budget =50	TimePFN	**0.235**	**0.322**	**0.190**	**0.235**	**0.746**	**0.468**	**0.429**	**0.450**	**0.096**	**0.218**	**0.438**	**0.429**	**0.324**	**0.359**	**0.419**	**0.418**	**0.195**	**0.276**
	iTransf.	0.278	0.360	0.237	0.278	0.801	0.499	0.513	0.479	0.145	0.275	0.838	0.617	0.410	0.422	0.884	0.608	0.268	0.337
	PatchTST	0.667	0.646	0.221	0.269	1.295	0.746	0.810	0.669	0.127	0.255	0.778	0.587	0.372	0.401	0.656	0.528	0.231	0.310
	DLinear	0.406	0.463	0.742	0.612	1.888	0.937	0.956	0.813	3.432	1.349	1.404	0.881	3.928	1.383	1.332	0.846	3.484	1.290
	FEDFormer	0.908	0.758	0.306	0.381	1.587	0.874	0.972	0.757	0.165	0.300	0.676	0.570	0.424	0.468	0.745	0.589	0.291	0.387
	Informer	1.226	0.896	0.464	0.511	1.714	0.901	0.887	0.783	1.470	1.007	1.172	0.819	2.045	1.093	1.003	0.745	1.590	0.995
	Autoformer	0.729	0.675	0.322	0.401	1.600	0.883	1.065	0.808	0.213	0.351	0.607	0.560	0.492	0.506	0.763	0.592	0.316	0.407
Budget =100	TimePFN	**0.221**	**0.309**	**0.187**	**0.232**	**0.644**	**0.424**	**0.351**	**0.383**	**0.083**	**0.205**	**0.441**	**0.429**	**0.322**	**0.356**	**0.412**	**0.411**	**0.196**	**0.273**
	iTransf.	0.253	0.337	0.220	0.263	0.740	0.468	0.369	0.387	0.138	0.268	0.728	0.574	0.401	0.418	0.816	0.586	0.260	0.331
	PatchTST	0.361	0.432	0.216	0.256	0.982	0.592	0.575	0.524	0.102	0.227	0.757	0.579	0.371	0.400	0.502	0.461	0.215	0.298
	DLinear	0.332	0.409	0.636	0.562	1.770	0.897	0.887	0.784	2.712	1.172	1.256	0.826	3.237	1.246	1.214	0.799	2.810	1.140
	FEDformer	0.597	0.598	0.264	0.344	1.350	0.775	0.951	0.752	0.158	0.291	0.562	0.513	0.362	0.407	0.724	0.572	0.290	0.387
	Informer	1.056	0.837	0.369	0.432	1.609	0.860	0.731	0.702	0.883	0.774	1.038	0.757	1.279	0.916	0.883	0.664	1.040	0.802
	Autoformer	0.468	0.519	0.251	0.330	1.344	0.773	0.960	0.762	0.187	0.321	0.550	0.524	0.379	0.422	0.704	0.554	0.258	0.351
Budget =500	TimePFN	**0.190**	**0.283**	**0.178**	**0.222**	**0.487**	**0.335**	**0.269**	**0.305**	0.083	0.203	**0.401**	**0.412**	**0.311**	**0.352**	**0.360**	**0.386**	**0.185**	**0.268**
	iTransf.	0.200	0.284	0.211	0.248	0.514	0.354	0.307	0.334	0.113	0.239	0.489	0.470	0.361	0.394	0.569	0.494	0.231	0.310
	PatchTST	0.236	0.320	0.210	0.246	0.740	0.455	0.321	0.353	**0.081**	**0.198**	0.596	0.515	0.358	0.392	0.369	0.386	0.190	0.275
	DLinear	0.235	0.328	0.335	0.394	1.312	0.727	0.622	0.656	0.655	0.551	0.749	0.609	1.098	0.712	0.817	0.621	0.870	0.626
	FEDformer	0.317	0.407	0.265	0.341	0.888	0.548	0.821	0.706	0.157	0.288	0.444	0.452	0.358	0.401	0.674	0.542	0.238	0.322
	Informer	0.869	0.760	0.320	0.393	1.411	0.774	0.318	0.385	0.699	0.694	0.913	0.713	1.311	0.940	0.704	0.595	1.121	0.803
	Autoformer	0.303	0.396	0.237	0.312	0.896	0.549	0.950	0.787	0.158	0.290	0.456	0.456	0.339	0.384	0.672	0.534	0.223	0.308
Budget = 1000	TimePFN	**0.173**	**0.268**	**0.175**	**0.219**	**0.452**	**0.310**	0.243	0.288	0.084	0.204	**0.405**	**0.415**	**0.304**	**0.351**	**0.344**	**0.378**	**0.180**	**0.262**
	iTransf.	0.184	0.271	0.206	0.242	0.469	0.324	0.276	0.309	0.100	0.223	0.433	0.436	0.336	0.379	0.464	0.444	0.211	0.294
	PatchTST	0.219	0.304	0.198	0.237	0.683	0.420	0.280	0.324	**0.082**	**0.200**	0.490	0.467	0.337	0.378	0.353	0.375	0.187	0.272
	DLinear	0.218	0.310	0.254	0.331	1.076	0.627	0.488	0.569	0.193	0.330	0.562	0.513	0.528	0.507	0.629	0.528	0.380	0.437
	FEDformer	0.284	0.379	0.269	0.341	0.806	0.486	0.545	0.546	0.157	0.287	0.402	0.435	0.341	0.383	0.436	0.456	0.228	0.312
	Informer	0.693	0.647	0.341	0.413	1.231	0.678	**0.229**	**0.294**	0.689	0.666	0.887	0.710	1.357	0.939	0.682	0.596	0.615	0.596
	Autoformer	0.270	0.367	0.239	0.314	0.787	0.492	0.926	0.742	0.156	0.285	0.427	0.442	0.341	0.383	0.617	0.521	0.218	0.301
Budget = All	TimePFN	**0.138**	**0.137**	**0.166**	**0.208**	**0.392**	**0.260**	0.203	0.219	0.100	0.223	0.402	0.417	**0.293**	**0.343**	0.392	0.402	0.180	0.262
	iTransf.	0.147	0.239	0.175	0.215	0.393	0.268	0.201	0.233	0.086	0.206	**0.387**	0.405	0.300	0.349	0.342	0.376	0.185	0.272
	PatchTST	0.185	0.267	0.177	0.218	0.517	0.334	0.222	0.267	**0.080**	**0.196**	0.392	**0.404**	**0.293**	**0.343**	**0.318**	**0.357**	**0.177**	**0.260**
	DLinear	0.195	0.278	0.341	0.412	0.690	0.432	0.286	0.375	0.101	0.237	0.400	0.412	0.357	0.406	0.344	0.371	0.195	0.293
	FEDformer	0.196	0.310	0.227	0.313	0.573	0.357	0.242	0.342	0.148	0.280	0.380	0.417	0.340	0.386	0.363	0.408	0.191	0.286
	Informer	0.327	0.413	0.455	0.481	0.735	0.409	**0.190**	**0.216**	0.921	0.774	0.930	0.763	2.928	1.349	0.623	0.559	0.396	0.474
	Autoformer	0.214	0.327	0.273	0.344	0.605	0.376	0.455	0.480	0.141	0.271	0.440	0.446	0.364	0.408	0.520	0.490	0.233	0.311
Avg Acc. Budgets	TimePFN	**0.191**	**0.264**	**0.179**	**0.223**	**0.544**	**0.359**	**0.299**	**0.329**	**0.089**	**0.211**	**0.417**	**0.420**	**0.311**	**0.352**	**0.385**	**0.399**	**0.187**	**0.268**
	iTransf.	0.212	0.298	0.210	0.249	0.583	0.383	0.333	0.348	0.116	0.242	0.575	0.500	0.362	0.392	0.615	0.502	0.231	0.309
	PatchTST	0.334	0.394	0.204	0.245	0.843	0.509	0.442	0.427	0.094	0.215	0.603	0.510	0.346	0.383	0.440	0.421	0.200	0.283
	DLinear	0.277	0.358	0.462	0.462	1.347	0.724	0.648	0.639	1.419	0.728	0.874	0.648	1.830	0.851	0.867	0.633	1.548	0.757
	FEDformer	0.460	0.490	0.266	0.344	1.041	0.608	0.706	0.621	0.157	0.289	0.493	0.477	0.365	0.409	0.588	0.513	0.248	0.339
	Informer	0.834	0.711	0.390	0.446	1.340	0.724	0.471	0.476	0.932	0.783	0.988	0.752	1.784	1.047	0.779	0.632	0.952	0.734
	Autoformer	0.397	0.457	0.264	0.340	1.046	0.615	0.871	0.716	0.171	0.304	0.496	0.486	0.383	0.421	0.655	0.538	0.250	0.336
	# of Variates	321		21		862		137		8		7		7		7		7	

Table 5: Results of *TimePFN* on various benchmarks, compared to baseline models. *TimePFN* has been fine-tuned using specified data budgets, with MSE and MAE scores reported. The best results are highlighted in bold, and both input and prediction lengths are set at 96. *TimePFN* demonstrates remarkable performance in budget-limited settings, as well as with the full dataset, particularly in scenarios involving a large number of variates.

dently. This case, as we described previously, corresponds to the case where $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$ with number of channels equaling to number of latent functions. We see in Table 4 that using generative coregionalization provides clear benefits.

E Extended Results

In addition to the budget scenarios presented in the main body, we also conducted experiments with data budgets of 100 and 1,000 to fully characterize our experimental results. Furthermore, the average accuracy across these data budgets is provided for reference. Table 5 showcases all these evaluations. In Table 6, we present the raw results of the univariate forecasting task for zero-shot forecasting.

Prediction Length		6		8		14		18		24		36		48		Average	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
TimePFN-96	Exchange	0.015	0.094	0.017	0.102	0.026	0.124	0.030	0.136	0.037	0.149	0.052	0.174	0.065	0.195	0.034	0.139
	Weather	0.020	1.006	0.023	1.068	0.034	1.280	0.041	1.411	0.051	1.582	0.072	1.885	0.087	2.108	0.046	1.477
	Traffic	0.388	0.489	0.397	0.496	0.409	0.506	0.408	0.499	0.413	0.499	0.436	0.516	0.449	0.520	0.414	0.503
	ECL	0.368	0.471	0.410	0.497	0.497	0.550	0.518	0.560	0.539	0.571	0.602	0.600	0.629	0.600	0.509	0.549
	ETTh1	0.017	0.101	0.020	0.109	0.026	0.126	0.030	0.134	0.034	0.143	0.041	0.156	0.045	0.163	0.030	0.133
	ETTh2	0.059	0.185	0.067	0.198	0.082	0.220	0.087	0.227	0.091	0.234	0.104	0.249	0.112	0.260	0.086	0.224
TimePFN-36	Exchange	0.012	0.087	0.014	0.094	0.020	0.112	0.024	0.122	0.030	0.134	0.041	0.155	0.052	0.174	0.027	0.125
	Weather $\times 10^2$	0.017	0.932	0.020	0.991	0.029	1.174	0.036	1.301	0.046	1.470	0.065	1.775	0.081	2.024	0.042	1.381
	Traffic	1.393	1.008	**1.528**	**1.051**	**1.644**	**1.084**	**1.520**	**1.031**	**1.403**	**0.988**	**1.538**	**1.039**	**1.495**	**1.027**	**1.503**	**1.032**
	ECL	0.585	0.621	**0.640**	0.649	**0.745**	**0.701**	**0.747**	**0.702**	**0.760**	**0.712**	**0.878**	**0.764**	**0.909**	**0.772**	**0.752**	**0.703**
	ETTh1	0.018	0.100	0.020	0.107	0.025	0.121	**0.028**	**0.128**	**0.032**	**0.137**	**0.040**	**0.153**	**0.045**	**0.164**	**0.029**	0.130
	ETTh2	0.100	0.241	**0.110**	0.253	**0.126**	**0.274**	**0.125**	**0.274**	**0.126**	**0.275**	**0.145**	**0.295**	**0.152**	**0.302**	**0.126**	**0.273**
ForecastPFN	Exchange	0.041	0.154	0.042	0.158	0.049	0.169	0.054	0.177	0.061	0.187	0.072	0.201	0.084	0.215	0.057	0.180
	Weather $\times 10^2$	0.062	1.668	0.065	1.719	0.074	1.865	0.080	1.952	0.089	2.073	0.103	2.278	0.115	2.443	0.084	1.999
	Traffic	4.690	1.779	4.712	1.790	4.572	1.765	4.428	1.724	4.348	1.698	4.504	1.735	4.394	1.703	4.521	1.742
	ECL	1.430	0.962	1.444	0.969	1.406	0.955	1.360	0.935	1.356	0.935	1.453	0.973	1.467	0.977	1.416	0.958
	ETTh1	0.085	0.216	0.087	0.220	0.093	0.228	0.097	0.232	0.104	0.239	0.119	0.256	0.131	0.270	0.102	0.237
	ETTh2	0.409	0.504	0.418	0.510	0.424	0.513	0.421	0.509	0.426	0.511	0.462	0.532	0.481	0.540	0.434	0.517
Chronos-Small	Exchange	0.026	0.072	0.048	0.081	0.079	0.104	0.020	0.107	0.059	0.124	**0.034**	0.141	0.075	0.165	0.049	0.113
	Weather $\times 10^2$	0.013	0.623	**0.014**	0.703	**0.023**	0.920	0.030	1.055	0.041	1.244	**0.058**	1.568	0.078	1.842	0.036	1.136
	Traffic	**1.298**	**0.819**	1.997	1.056	3.738	1.530	4.063	1.642	3.545	1.502	3.434	1.482	3.646	1.519	3.103	1.364
	ECL	**0.473**	**0.488**	0.698	**0.601**	1.313	0.856	1.443	0.914	1.310	0.865	1.371	0.893	1.458	0.931	1.152	0.792
	ETTh1	0.045	0.114	0.044	0.121	0.062	0.151	0.065	0.159	0.065	0.168	0.073	0.184	0.076	0.194	0.061	0.155
	ETTh2	**0.089**	**0.188**	0.134	**0.238**	0.227	0.337	0.251	0.365	0.235	0.358	0.250	0.374	0.266	0.393	0.207	0.321
SeasonalNaive	Exchange	0.015	0.096	0.016	0.100	0.021	0.114	0.025	0.124	0.030	0.135	0.039	0.154	0.050	0.172	0.028	0.128
	Weather $\times 10^2$	0.021	0.907	0.023	0.965	0.031	1.137	0.039	1.278	0.048	1.445	0.067	1.740	0.084	1.989	0.045	1.352
	Traffic	4.354	1.850	4.581	1.891	5.263	2.016	4.416	1.784	3.756	1.614	4.104	1.691	3.631	1.548	4.301	1.771
	ECL	1.427	0.962	1.523	0.994	1.810	1.092	1.590	1.004	1.427	0.942	1.600	1.001	1.533	0.973	1.559	0.995
	ETTh1	0.027	0.126	0.029	0.131	0.035	0.145	0.037	0.149	0.040	0.156	0.049	0.171	0.055	0.181	0.039	0.151
	ETTh2	0.272	0.394	0.283	0.405	0.313	0.437	0.278	0.409	0.254	0.390	0.279	0.413	0.273	0.406	0.279	0.408
Naive	Exchange	**0.008**	**0.064**	**0.010**	**0.073**	**0.015**	**0.093**	**0.019**	**0.104**	**0.024**	**0.118**	**0.034**	**0.140**	**0.045**	**0.160**	**0.022**	**0.107**
	Weather $\times 10^2$	**0.011**	**0.598**	**0.014**	**0.685**	**0.023**	**0.910**	**0.029**	**1.044**	**0.038**	**1.232**	**0.058**	**1.561**	**0.075**	**1.834**	**0.035**	**1.123**
	Traffic	1.759	1.041	2.495	1.263	4.090	1.661	4.245	1.716	3.524	1.504	3.622	1.536	3.574	1.517	3.330	1.463
	ECL	0.586	0.560	0.827	0.672	1.400	0.904	1.479	0.941	1.309	0.874	1.406	0.914	1.471	0.938	1.211	0.829
	ETTh1	**0.014**	**0.084**	**0.018**	**0.095**	**0.027**	**0.120**	0.031	0.131	0.034	0.139	0.043	0.157	0.050	0.171	0.031	**0.128**
	ETTh2	0.114	0.226	0.157	0.272	0.240	0.357	0.256	0.376	0.229	0.357	0.248	0.377	0.259	0.390	0.215	0.336
Mean	Exchange	0.027	0.131	0.028	0.135	0.033	0.146	0.037	0.152	0.042	0.161	0.052	0.177	0.062	0.192	0.040	0.156
	Weather $\times 10^2$	0.047	1.546	0.050	1.599	0.059	1.775	0.064	1.840	0.074	1.966	0.089	2.178	0.101	2.345	0.069	1.893
	Traffic	2.293	1.332	2.350	1.343	2.233	1.287	2.049	1.221	1.920	1.183	2.078	1.234	1.955	1.192	2.125	1.256
	ECL	0.923	0.793	0.955	0.805	0.960	0.805	0.929	0.790	0.923	0.788	1.025	0.828	1.029	0.827	0.963	0.805
	ETTh1	0.031	0.135	0.033	0.138	0.036	0.146	0.038	0.151	0.041	0.157	0.048	0.170	0.052	0.178	0.040	0.154
	ETTh2	0.161	0.314	0.166	0.320	0.167	0.321	0.162	0.315	0.162	0.314	0.179	0.330	0.182	0.333	0.168	0.321

Table 6: Zero-shot results of TimePFN on univariate time-series forecasting with input length = 36. TimePFN-96 has input length of 96. All other baselines have input length 36. Meta-N-Beats is not included as it is not our implementation.

E.1 Multivariate Forecasting

As shown in Table 5, *TimePFN* consistently achieves the best results with a data budget of 100 and significantly outperforms all other models with a budget of 1,000, leading in 7 out of 9 datasets. *TimePFN* excels particularly in datasets with a multivariate nature. Consider that PatchTST [19] assumes channel independence, whereas iTransformer [16] treats each variate as a token, demonstrating extreme channel dependence. In the full budget scenario, where the entire dataset is utilized, the difference in forecasting performance between iTransformer and PatchTST is revealing, particularly in detecting datasets with high inter-channel dependencies. For instance, in the ECL and Traffic datasets, which have a large number of variates (which does not mean high channel dependence by itself), iTransformer shows superior forecasting performance compared to PatchTST. Conversely, in the ETT datasets, PatchTST performs comparatively better. Extrapolating from there, we realize that *TimePFN* excels in datasets with a high multivariate nature, even in full budget scenarios, and also yields good and competitive performance in datasets with comparatively low multivariate characteristic in full budget scenarios. With limited budgets, we see that *TimePFN* is the leading model among the baselines.

E.2 Univariate Forecasting

Although *TimePFN* is specifically designed for multivariate time series forecasting, we also assessed its performance in zero-shot univariate forecasting, compared to similar models. See Table 6 for

more details. On average, *TimePFN*-36 is the most successful model among other models, and uniformly better than all other deep-learning based baselines in our setting. Generally, Chronos-small [2] outperforms *TimePFN*-36 with shorter prediction lengths, while TimePFN-36 excels at longer prediction lengths, outperforming the other models. This outcome is expected, as *TimePFN* is specifically trained to handle an input length of 96 and predict the same distance ahead. For these comparisons, we trimmed *TimePFN*'s predictions to match the given prediction lengths. Given *TimePFN*'s focus on longer prediction horizons, it's no surprise that Chronos-small performs better at shorter lengths. For *TimePFN*-36, we padded the first 60 sequences of the 96-sequence input with the average of a 36-sequence input to minimize distribution shift. We also included results for *TimePFN*-96, which uses the full 96-sequence input length without padding, to demonstrate our model's complete performance.

F Datasets

As datasets, we used 9 benchmark datasets which are commonly used in multivariate time-series forecasting. These consist of four ETT datasets [28] (ETTh1, ETTh2, ETTm1, ETTm2), ECL (Electricity Consuming Load), Exchange, Traffic, Weather and Solar Energy. Except for the Solar Energy datasets, the others are benchmarked by [25], while the Solar is introduced by [13]. We splitted the training, validation and test sets in a chronological way deterministically, following [19, 16]. We will briefly explain the features of these datasets in this section.

ETT Datasets. The abbreviation ETT refers to Electricity Transformer Temperature [28]. All ETT datasets consist of seven variates. The ETTh1 and ETTh2 datasets are sampled hourly, while the ETTm1 and ETTm2 datasets are sampled every 15 minutes. Specifically, the ETTh datasets contain 8545, 2881, and 2881 data points in the training, validation, and test sets, respectively. In contrast, the ETTm datasets comprise 34465, 11521, and 11521 data points in the training, validation, and test sets, respectively [16].

ECL Dataset. The abbreviation ECL refers to the electricity consumption load of 321 users [25]. It is recorded in hourly intervals, resulting in a dataset with 321 variates. The ECL dataset contains 18317, 2633, and 5261 data points in the training, validation, and test sets, respectively [16].

Exchange Dataset. This dataset provides daily exchange rates for eight countries [25], comprising eight variates. The Exchange dataset includes 5120, 665, and 1422 data points in the training, validation, and test sets, respectively [16]. Some works, such as PatchTST [19], avoid using this dataset as a benchmark because simple naive predictions (using the last observed value) often outperform more complex methods. However, for completeness, we have included it in our analysis.

Traffic Dataset. This dataset includes hourly road occupancy rates from 862 locations [25], resulting in 862 variates. The traffic dataset contains 12185, 1757, and 3509 data points in the training, validation, and test sets, respectively [16]. It is by far the most high-dimensional dataset in our evaluation.

Weather Dataset. This dataset includes 21 meteorological factors collected every 10 minutes [25], resulting in 21 variates. The weather dataset contains 36792, 5271, and 10540 data points in the training, validation, and test sets, respectively [16].

Solar-Energy Dataset. This dataset includes power production values from 137 solar power plants, sampled every 10 minutes [13], resulting in 137 variables. The solar energy dataset contains 36601, 5161, and 10417 data points in the training, validation, and test sets, respectively [16].

G Additional Ablations

In addition to the ablation studies we provided, we include three additional case studies, with one focusing on the performance of the architecture of *TimePFN* when it is not pretrained synthetic data, and the second one focuses on the performace of iTransformer [16] when it is used as the PFN backbone in zero-shot setting. In the third case study, we evaluate this iTransformer-PFN compared to iTransformer in various data budgets to demonstrate the generality of the framework of large-scale synthetic training and to demonstrate the architectural novelties of *TimePFN*. Table 7, Table 8 and Table 9 contain the results of those ablation studies, respectively.

	Dataset	ECL		Weather		Traffic		Solar-Energy		Exchange		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
50	TimePFN	**0.235**	**0.322**	**0.190**	**0.235**	**0.746**	**0.468**	**0.429**	0.450	**0.096**	**0.218**	**0.438**	**0.429**	**0.324**	**0.359**	**0.419**	**0.418**	**0.195**	**0.276**
	TimePFN-w/o-s	0.314	0.391	0.213	0.255	0.966	0.580	0.448	**0.440**	0.110	0.237	0.520	0.482	0.360	0.392	0.468	0.445	0.234	0.312
100	TimePFN	**0.221**	**0.309**	**0.187**	**0.232**	**0.644**	**0.424**	0.351	0.383	**0.083**	**0.205**	**0.441**	**0.429**	**0.322**	**0.356**	0.412	**0.411**	**0.196**	**0.273**
	TimePFN-w/o-s	0.259	0.344	0.221	0.257	1.008	0.596	**0.324**	**0.347**	0.104	0.230	0.505	0.475	0.360	0.391	**0.404**	**0.411**	0.231	0.308
500	TimePFN	**0.190**	**0.283**	**0.178**	**0.222**	**0.487**	**0.335**	**0.269**	**0.305**	**0.083**	**0.203**	**0.401**	**0.412**	**0.311**	**0.352**	0.360	0.386	**0.185**	**0.268**
	TimePFN-w/o-s	0.220	0.311	0.191	0.235	0.914	0.559	0.303	0.318	0.105	0.232	0.423	0.431	0.354	0.387	**0.357**	**0.383**	0.229	0.307
1000	TimePFN	**0.173**	**0.268**	**0.175**	**0.219**	**0.452**	**0.310**	**0.243**	**0.288**	**0.084**	**0.204**	**0.405**	**0.415**	**0.304**	**0.351**	0.344	0.378	**0.180**	**0.262**
	TimePFN-w/o-s	0.187	0.285	0.182	0.224	0.907	0.556	0.278	0.302	0.109	0.233	0.409	0.425	0.352	0.387	**0.341**	**0.374**	0.206	0.289
All	TimePFN	0.138	0.137	**0.166**	**0.208**	**0.392**	**0.260**	0.203	0.219	**0.100**	**0.223**	**0.402**	**0.417**	**0.293**	**0.343**	**0.392**	**0.402**	**0.180**	**0.262**
	TimePFN-w/o-s	**0.137**	**0.136**	0.214	0.269	1.408	0.800	**0.196**	**0.210**	0.101	0.225	0.412	0.423	0.352	0.386	0.437	0.414	0.229	0.307

Table 7: In TimePFN-w/o-s, we do not pretrain the TimePFN architecture. Instead, we directly train the architecture with given data budgets and compare it with the pretrained and fine-tuned TimePFN. Both the sequence lengths and the forecasting length are set to 96.

	Dataset	ECL		Weather		Traffic		Solar-Energy		Exchange		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
z.s.	TimePFN	**0.315**	**0.383**	**0.209**	**0.255**	**1.108**	**0.613**	**0.941**	**0.730**	**0.105**	**0.229**	**0.453**	**0.439**	0.328	**0.362**	**0.637**	**0.512**	**0.212**	**0.291**
	TimePFN-w/o-C	0.653	0.637	0.221	0.271	1.287	0.757	1.197	0.829	0.111	0.237	0.608	0.517	0.338	0.374	0.771	0.565	0.224	0.307
	PatchTST-PFN	0.470	0.522	0.212	0.262	1.172	0.702	1.014	0.787	0.108	0.231	0.554	0.501	**0.322**	0.366	0.746	0.560	0.215	0.301
	iTrans.-PFN	0.609	0.605	0.213	0.261	1.321	0.761	1.075	0.791	0.111	0.235	0.522	0.481	0.327	0.367	0.765	0.563	0.220	0.305

Table 8: Extended ablation results involving iTransformer-PFN. In iTransformer-PFN, we pre-train an iTransformer model to assess the suitability of our architecture for PFNs, using a sequence length and forecasting length of 96. In TimePFN-w/o-Convolution, we remove the convolutional operator typically applied to the initial input variables. In PatchTST-PFN, we train a PatchTST model to evaluate the importance of channel mixing and the suitability of our architecture for PFNs, also using a sequence length and forecasting length of 96. We evaluate the zero-shot performance.

Pretraining TimePFN with LMC-Synth. To better understand the impact of synthetic training in *TimePFN*, we removed the synthetic training component and directly trained the architecture, which we referred to as *TimePFN*-w/o-synthetic. As shown in Table 7, the forecasting performance of *TimePFN* is significantly better than that of *TimePFN*-w/o-synthetic, demonstrating its contribution in both full budget and limited budget settings.

iTransformer as PFN. To evaluate the performance of other architectures with prior-data-fitting, we trained an iTransformer architecture with LMC-Synth in addition to PatchTST-PFN. As shown in Table 8, compared to other variations, *TimePFN* demonstrates significantly better zero-shot forecasting capability, with uniformly better results than those of the competing architectures. This supports our architectural design principles, involving 1D convolutions and channel-mixing.

iTransformer-PFN with Data Budgets. To demonstrate the behavior of another architecture with synthetic training, we pre-trained the iTransformer [16] with data generated by LMC-Synth and fine-tuned it using specified data budgets. As shown in Table 9, synthetic pre-training improves the performance of the model in most cases, demonstrating the generality of the framework. However, the contribution is limited when using the entire dataset, in contrast to those instances with *TimePFN*, thereby highlighting *TimePFN*'s superior performance.

H Baseline Details

In addition to the main body, we would like to elaborate on how we aggregated the forecasting results from Chronos-small [2] and ForecastPFN [5]. We chose Chronos-small because its parameter size is somewhat closer to *TimePFN*'s compared to the larger variations of Chronos. We opted not to use Chronos-Tiny to maintain a stronger baseline. *TimePFN* has approximately 8.5 million parameters, whereas Chronos-small has 46 million parameters, significantly exceeding the parameter count of *TimePFN*. As Chronos is a probabilistic model, we performed inference on each time-series data point three times with Chronos and averaged the results to obtain the final point forecast, except for the Exchange dataset. Our evaluation of Chronos on the Exchange dataset yielded unstable results with three inference iterations; therefore, we conducted five runs for this dataset. We aggregated the ForecastPFN results using their published model weights.

	Dataset	ECL		Weather		Traffic		Solar-Energy		Exchange		ETTh1		ETTh2		ETTm1		ETTm2	
	Models	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
50	iTrans.-PFN	**0.255**	**0.344**	**0.215**	**0.250**	1.292	0.725	**0.392**	**0.412**	**0.102**	**0.230**	**0.491**	**0.465**	**0.370**	**0.393**	**0.466**	**0.440**	**0.217**	**0.301**
	iTransformer	0.278	0.360	0.237	0.278	**0.801**	**0.499**	0.513	0.479	0.145	0.275	0.838	0.617	0.410	0.422	0.884	0.608	0.268	0.337
100	iTrans.-PFN	**0.227**	**0.318**	**0.213**	**0.249**	1.216	0.702	**0.336**	**0.355**	**0.101**	**0.226**	**0.478**	**0.455**	**0.370**	**0.393**	**0.430**	**0.424**	**0.204**	**0.287**
	iTransformer	0.253	0.337	0.220	0.263	**0.740**	**0.468**	0.369	0.387	0.138	0.268	0.728	0.574	0.401	0.418	0.816	0.586	0.260	0.331
500	iTrans.-PFN	**0.184**	**0.276**	**0.200**	**0.241**	1.201	0.698	0.309	**0.333**	**0.096**	**0.223**	**0.452**	**0.444**	**0.319**	**0.362**	**0.375**	**0.394**	**0.199**	**0.284**
	iTransformer	0.200	0.284	0.211	0.248	**0.514**	**0.354**	**0.307**	0.334	0.113	0.239	0.489	0.470	0.361	0.394	0.569	0.494	0.231	0.310
1000	iTrans.-PFN	**0.170**	**0.263**	**0.195**	**0.238**	1.239	0.696	0.288	0.308	**0.097**	**0.223**	**0.431**	**0.432**	**0.304**	**0.352**	**0.365**	**0.391**	**0.196**	**0.279**
	iTransformer	0.184	0.271	0.206	0.242	**0.469**	**0.324**	**0.276**	**0.309**	0.100	**0.223**	0.433	0.436	0.336	0.379	0.464	0.444	0.211	0.294
All	iTrans.-PFN	**0.147**	0.240	0.209	0.259	1.408	0.800	0.231	0.262	0.104	0.233	0.424	0.428	0.350	0.385	0.666	0.533	0.223	0.302
	iTrans.	**0.147**	**0.239**	**0.175**	**0.215**	**0.393**	**0.268**	**0.201**	**0.233**	**0.086**	**0.206**	**0.387**	**0.405**	**0.300**	**0.349**	**0.342**	**0.376**	**0.185**	**0.272**

Table 9: In the iTrans.-PFN, we pre-trained an iTransformer model on a large-scale synthetic dataset. The data budget evaluations involve fine-tuning the model using these data budgets. For the iTransformer evaluations, we utilized the official hyperparameters of the model. Both the sequence lengths and the forecasting lengths are set to 96.

I Implementation Details

We implemented *TimePFN* entirely in PyTorch. We optimized the pretraining task using a synthetic dataset generated with LMC-Synth, employing the Adam optimizer [12] and adhering to a one-cycle learning rate policy with a maximum learning rate of $lr = 0.0005$ [22]. In the few-shot evaluations, we fine-tuned the *TimePFN* with maximum $lr = 0.0002$ using AdamW optimizer [17] with one-cycle learning rate policy. In training *TimePFN* with synthetic dataset, we observed that making model see the independently generated channels first, corresponding to the case with $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$, then introducing the inter-channel dependent data, significanly improves the learning speed. The explanation is simple, with the case with $C_i(\boldsymbol{t}) = l_i(\boldsymbol{t})$, the model sees much more time-series patterns, as the time-series channels are all independently generated by Gaussian processes. Thus, after the model learns to make channel-independent decisions, we introduced the channel-dependent data, similar to curriculum learning scenario [4]. Moreover, while training on synthetic data, we added a multiplicative Gaussian noise to each MTS data point as a regularization, with $\sigma = 0.1, mean = 1$. In the end, *TimePFN* is trained on 1.5 million MTS data points with 160 channels.

In *TimePFN*, we used the token embedding dimension of 256, and the latent space dimension of 1024, while the feed-forward network dimension is set to 512. We did not do any hyperparameter tuning and chose those values from checking similar works such as [19, 16]. In fine-tuning, we always used the same learning rate with the same number of epochs accross different datasets (0.0002 and 8 epochs). Thus, we run the evaluations once. While doing all those, we fixed the seed to a random value of 2023.

Throughout the experiments, we used a single L40S GPU. In addition the GPU, we had access to 128 GB of RAM and a 32-core CPU, which facilitated the acceleration of synthetic time-series data generation. Our codebase is developed based on [16]. Overall, we used approximately 300 GPU hours, as we conducted benchmarks not only for our own model but also for many others. We are providing the full source code for *TimePFN*, including the synthetic data generation, architecture, and the training and evaluation scripts. Furthermore, we are providing the model weights of *TimePFN*.

J Visualizations

Visualizations for LMC-Synth. We provide visualizations for the multivariate synthetic data generated by LMC-Synth. For clarity, we have limited the number of channels to 5 and the sequence length to 96. Figures 3-6 showcase the MTS data generated using LMC-Synth.

The Forecasts of TimePFN. We provide forecasts from *TimePFN* under various data budgets and datasets, including zero-shot and full-data scenarios. Figures 7-18 display the forecasts of *TimePFN* in these settings. As shown, with an increasing data budget, *TimePFN*'s forecasts align more closely with the ground truth.

Optionally include extra information (complete proofs, additional experiments and plots) in the appendix.

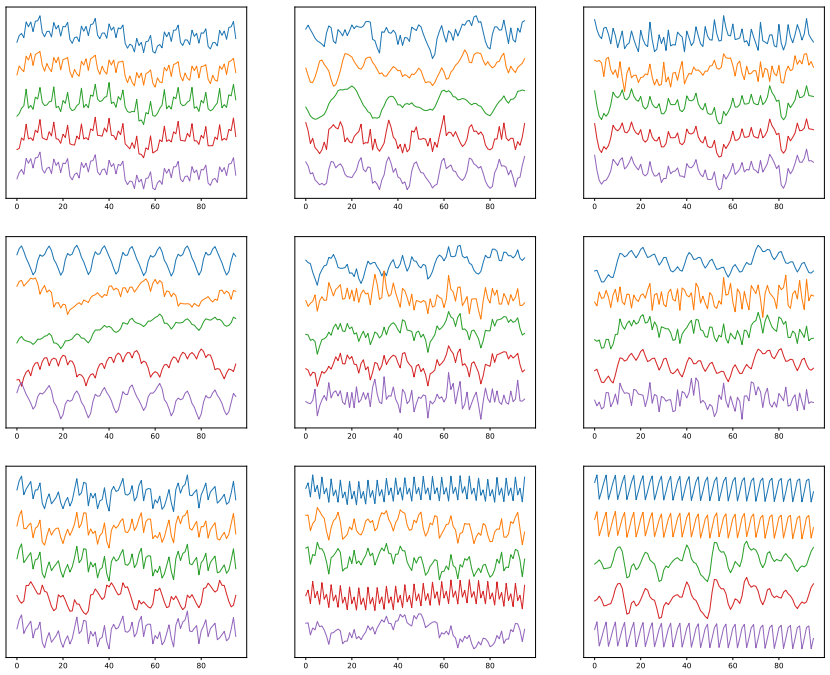


Figure 2: Examples of synthetic multivariate time time-series data generated by LMC-Synth. For the ease of understanding, we took C=5 and sequence lenght = 96. Dirichlet concentration parameter controls the diversity of variates from one another.

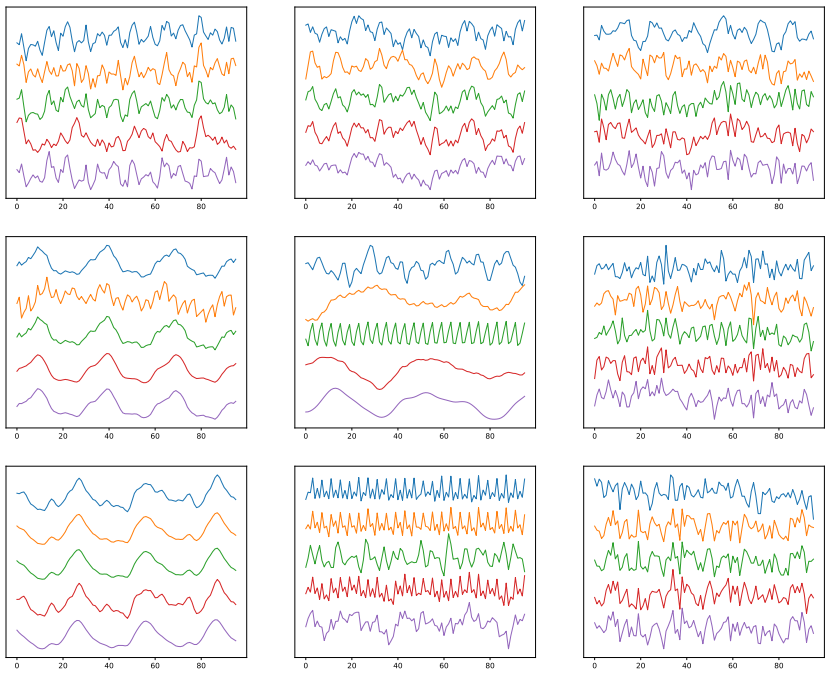


Figure 3: Examples of synthetic multivariate time time-series data generated by LMC-Synth. For the ease of understanding, we took C=5 and sequence lenght = 96. Dirichlet concentration parameter controls the diversity of variates from one another.

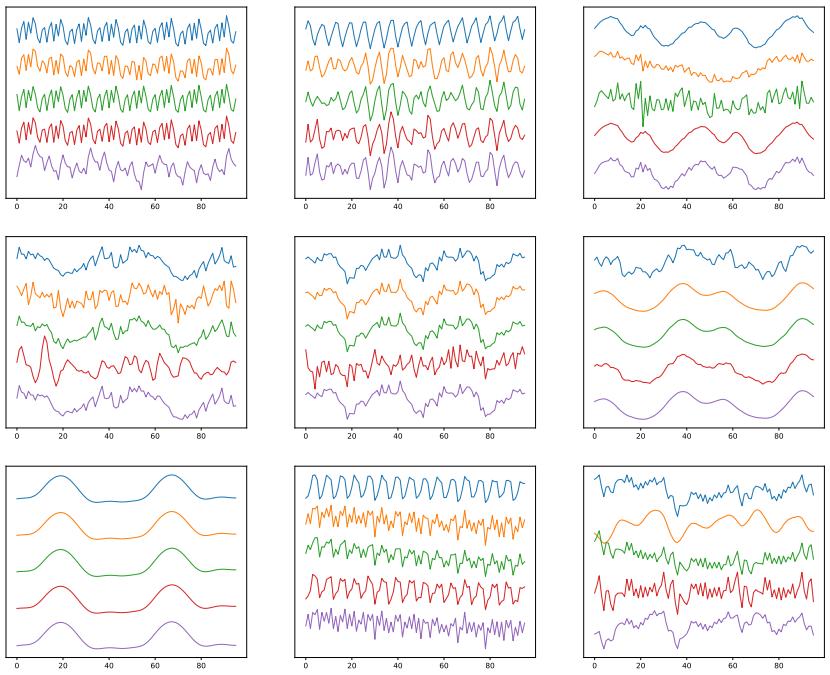


Figure 4: Examples of synthetic multivariate time time-series data generated by LMC-Synth. For the ease of understanding, we took C=5 and sequence lenght = 96. Dirichlet concentration parameter controls the diversity of variates from one another.

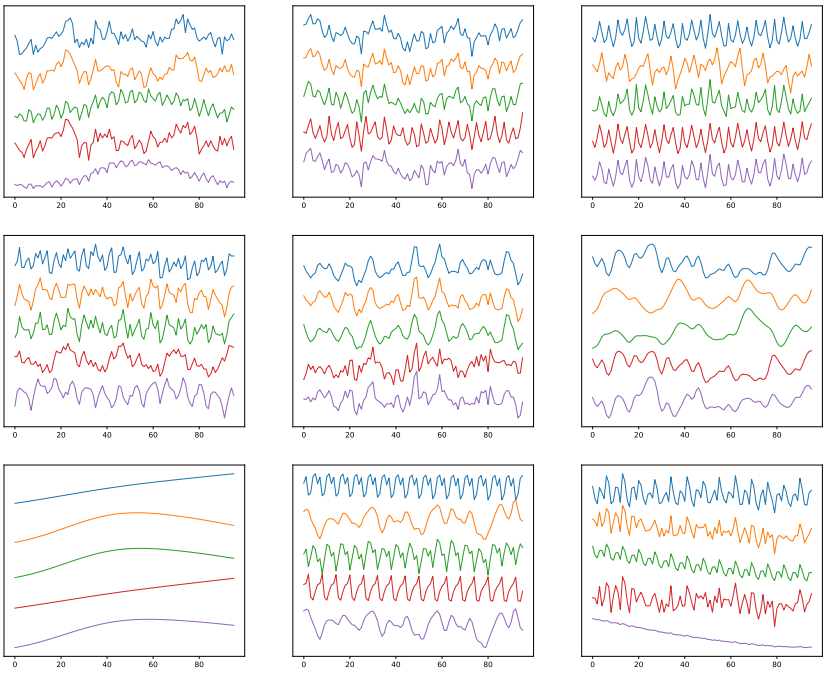
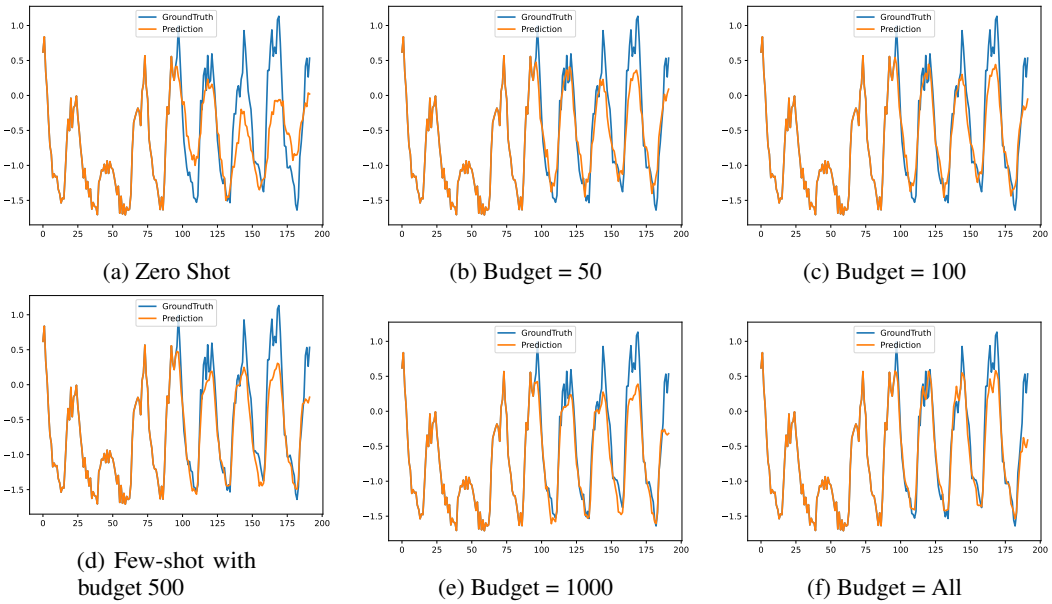


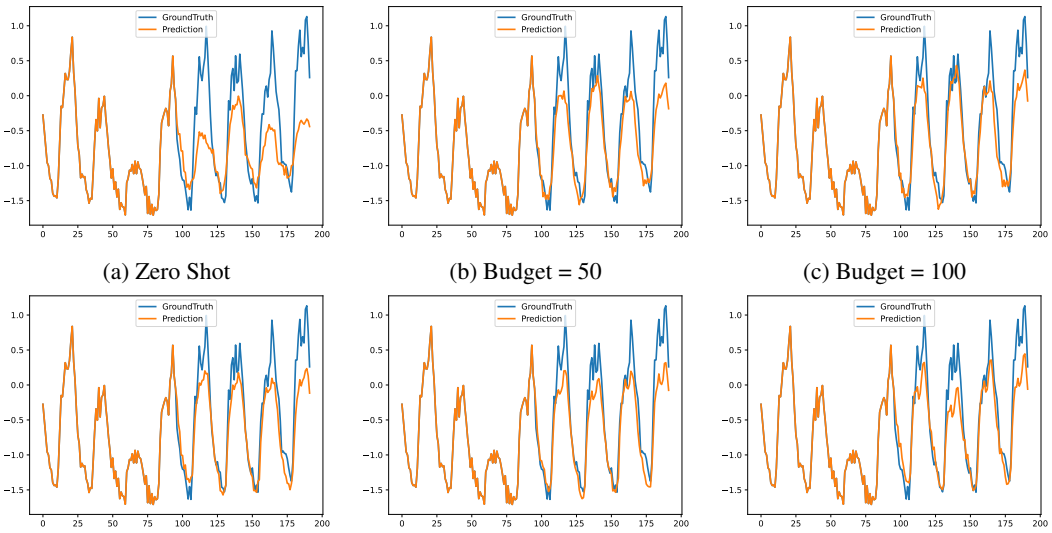
Figure 5: Examples of synthetic multivariate time time-series data generated by LMC-Synth. For the ease of understanding, we took C=5 and sequence lenght = 96. Dirichlet concentration parameter controls the diversity of variates from one another.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Few-shot with budget 500 (e) Budget = 1000 (f) Budget = All

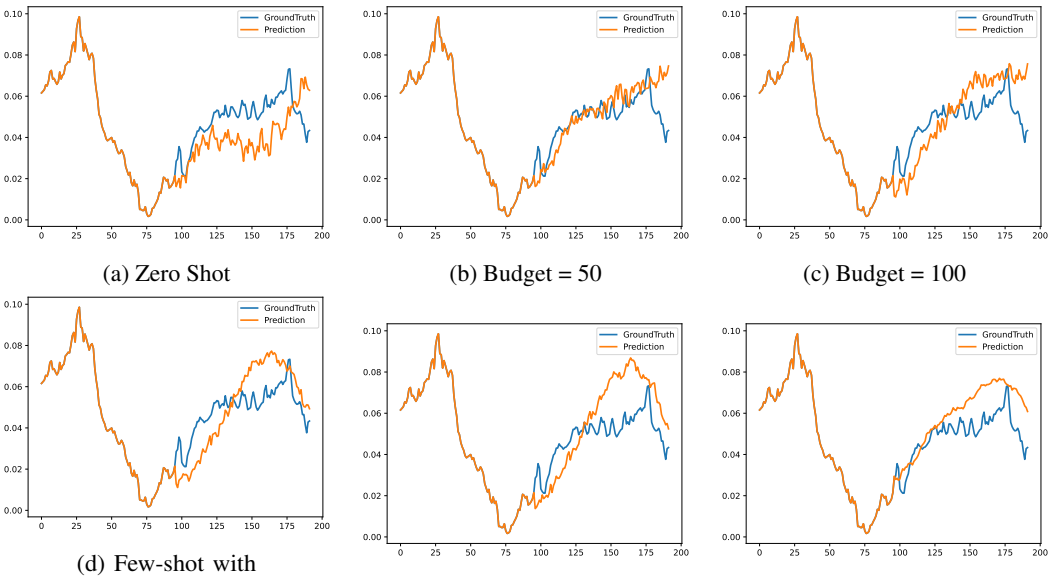
Figure 6: The forecasts of TimePFN with various data budgets on ECL dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Budget = 500 (e) Budget = 1000 (f) Budget = All

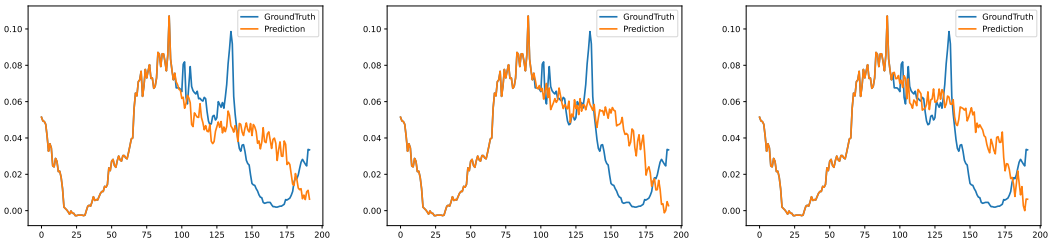
Figure 7: The forecasts of TimePFN with various data budgets on ECL dataset.



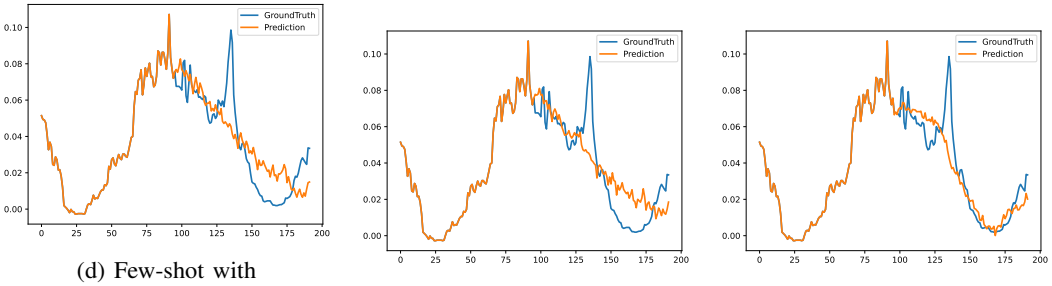
(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Few-shot with budget 500 (e) Budget = 1000 (f) Budget = All

Figure 8: The forecasts of TimePFN with various data budgets on weather dataset.

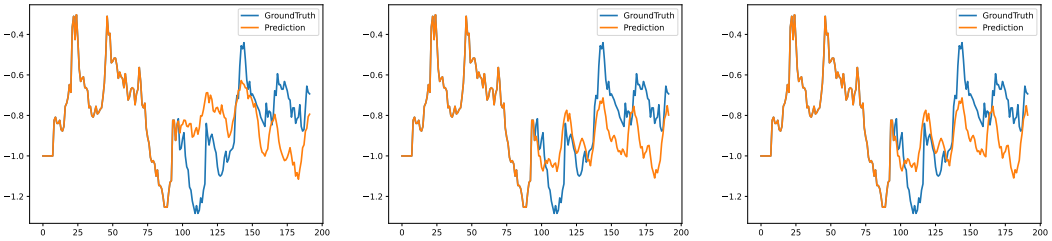


(a) Zero Shot (b) Budget = 50 (c) Budget = 100

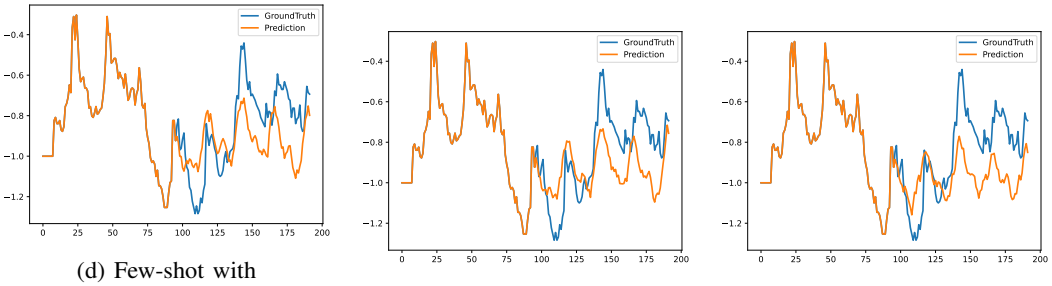


(d) Few-shot with budget 500 (e) Budget = 1000 (f) Budget = All

Figure 9: The forecasts of TimePFN with various data budgets on weather dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100



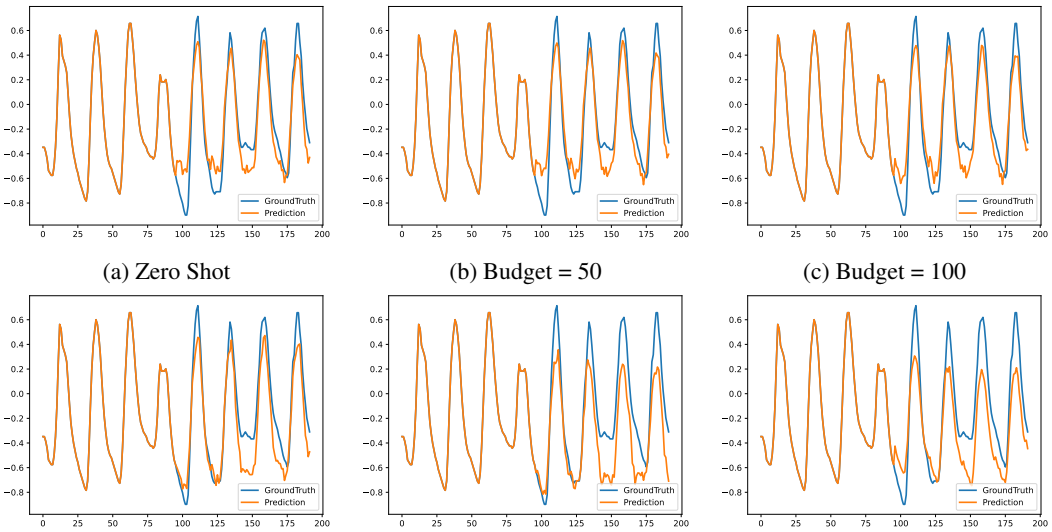
(d) Few-shot with budget 500 (e) Budget = 1000 (f) Budget = All

Figure 10: The forecasts of TimePFN with various data budgets on ETTh1 dataset.

(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Few-shot with budget 500 (e) Budget = 1000 (f) Budget = All

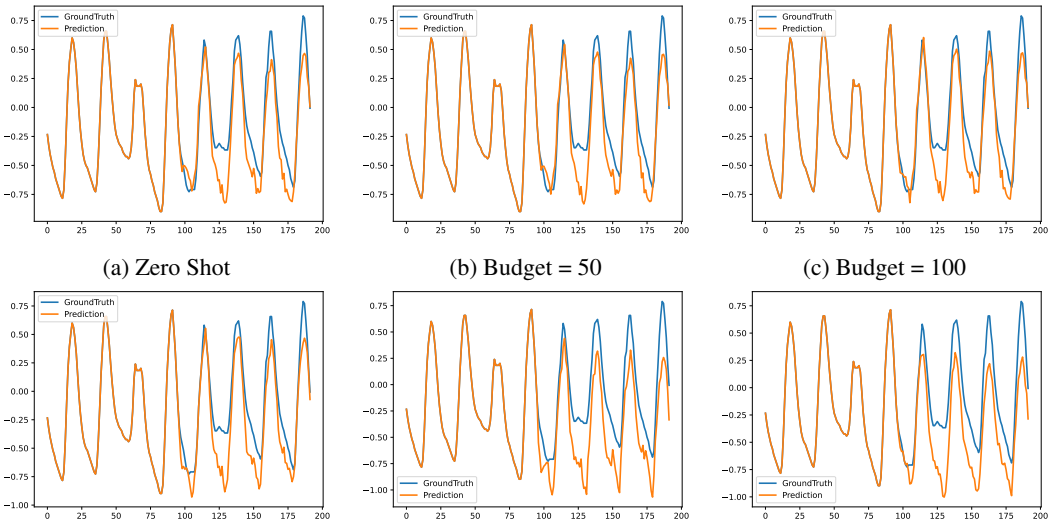
Figure 11: The forecasts of TimePFN with various data budgets on ETTh1 dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Budget = 500 (e) Budget = 1000 (f) Budget = All

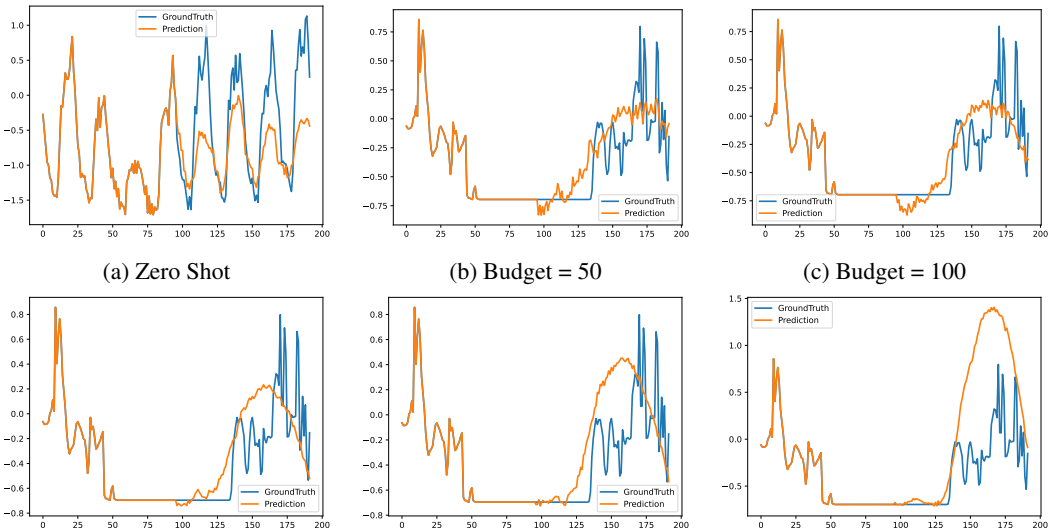
Figure 12: The forecasts of TimePFN with various data budgets on ETTh2 dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Budget = 500 (e) Budget = 1000 (f) Budget = All

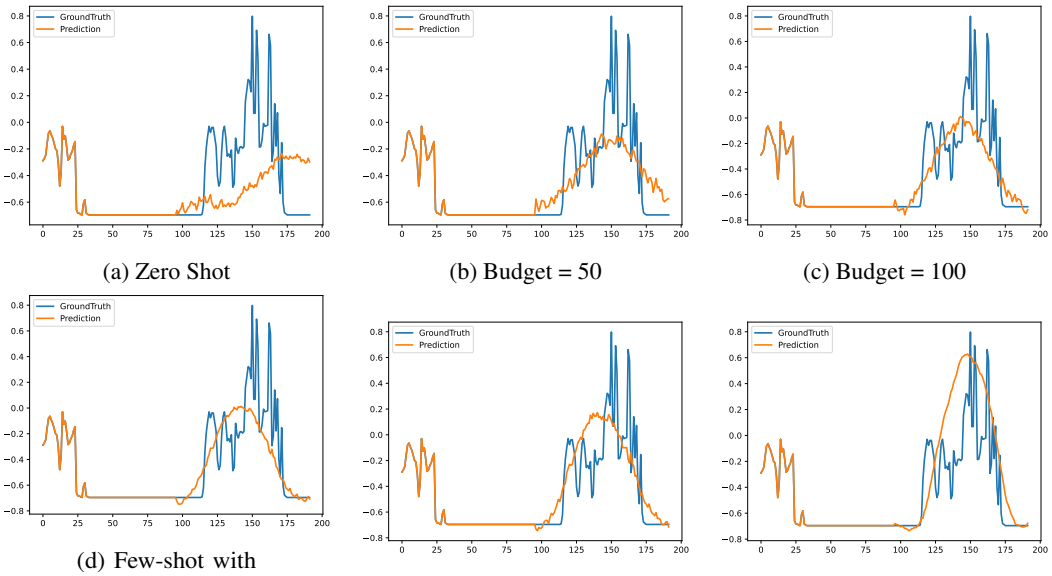
Figure 13: The forecasts of TimePFN with various data budgets on ETTh2 dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Budget = 500 (e) Budget = 1000 (f) Budget = All

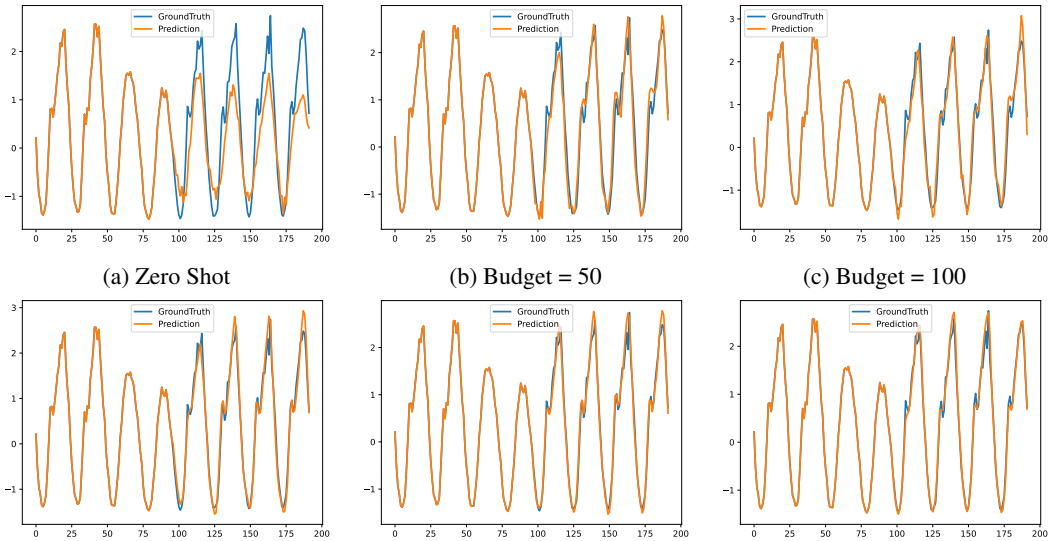
Figure 14: The forecasts of TimePFN with various data budgets on Solar dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Few-shot with budget 500 (e) Budget = 1000 (f) Budget = All

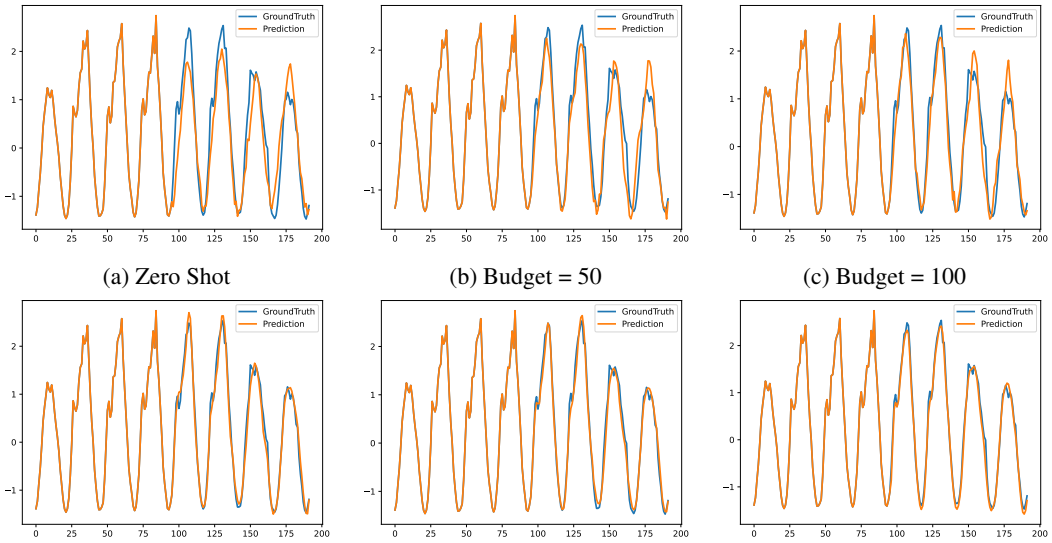
Figure 15: The forecasts of TimePFN with various data budgets on Solar dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Budget = 500 (e) Budget = 1000 (f) Budget = All

Figure 16: The forecasts of TimePFN with various data budgets on traffic dataset.



(a) Zero Shot (b) Budget = 50 (c) Budget = 100

(d) Budget = 500 (e) Budget = 1000 (f) Budget = All

Figure 17: The forecasts of TimePFN with various data budgets on traffic dataset.