Likelihood-Aligned Forecast Networks

Denizalp Goktas, Gerardo Riaño-Briceño, Amy Greenwald
Simulacrum
New York City, NY, USA
{deni, gerardo, amy}@smlcrm.com

Abstract

Foundation models have transformed natural language processing and computer vision. A rapidly growing literature on time-series foundation models (TSFMs) seeks to replicate this success in forecasting. TSFMs take as input a context, which consists of past time-series variables, and possibly covariate time-series data, and output predicted future values and/or distributions over the variables. Most TSFMs are formulated as neural networks trained via supervised learning, with virtually all employing transformer architectures. Despite the success of transformers in other disciplines, these architectures suffer from slow performance in time-series domains because of their high computational demands; moreover, they often fail to leverage long-range time and cross-feature data dependencies. In this paper, we investigate Likelihood-Aligned Forecast Networks (LAFNs), which output both point and probabilistic forecasts, and as such minimize a loss function that incorporates both Euclidean loss for the point prediction and likelihood loss for the probabilistic prediction. Our LAFN network architecture comprises a hypernetwork that generates weights for a target network, the latter of which captures strong temporal and cross-feature biases. By decoupling the forecasting model into hyper and target networks, LAFNs harness the power of foundation models, in their adaptability to a wide variety of downstream tasks, and, at the same time, leverage the inductive strengths of classic approaches to time-series prediction.

1 Introduction

Foundation models have revolutionized natural language processing and computer vision, establishing a new paradigm where large-scale pretraining yields broadly transferable representations for zero-shot tasks. Motivated by these successes, a rapidly expanding literature has emerged around time-series foundation models (TSFMs), forecasting models designed to generalize across diverse forecasting tasks. TSFMs typically take as input a historical context of past observations, along with optional covariates, and output forecasts or predictive distributions for future values. While transformer-based architectures dominate this space, these architectures suffer from slow performance in time-series domains because of their high computational demands; moreover, they often fail to leverage long-range time and cross-feature data dependencies. Consequently, as observed by practitioners (e.g., [1]), simple statistical models (e.g., ARIMA [2], SVR [3, 4]) with well-chosen hyperparameters have been seen to outperform large transformers in forecasting tasks where forecasting latency is paramount, as are long-range time and cross-feature data dependencies.[1]

This observation suggests a potential approach to building computationally efficient TSFMs that can model long-range and cross-feature dependencies, namely to train a network that, based on the input historical context, outputs parameters for a simple statistical model. Generalizing this insight further, in this paper, we study Likelihood-Aligned Forecast Networks (LAFNs), a small

[1] We include additional related works in Section F.

Preprint. Under review.

metanetwork [5, 6] that consists of a hypernetwork, which learns representations of time-series data and generates weights (i.e., hyperparameters) for a lightweight target network that captures strong inductive temporal and cross-feature biases. By decoupling the forecasting model into hyper and target networks, LAFNs are designed to harness the power of foundation models, in their adaptability to a wide variety of downstream tasks, and, at the same time, leverage the inductive strengths of classic approaches to time-series prediction.

In this work, we show that LAFNs are capable of generating forecasts on par with transformer-based TSFMs, while using only a fraction of the parameter count (460,000 parameters), and training exclusively on synthetic multivariate time-series data, as large, real-world multivariate time-series corpora are few and far between. We generate these synthetic datasets through the Cholesky decomposition of structured Gram matrices, producing controllable temporal and cross-feature patterns. Importantly, our network provides both point and probabilistic forecasts, and is trained with a novel loss function that aligns the two forecasts and leads to the extraction of more generalizable patterns from the training data. We believe that together, these design choices suggest a scalable path toward universal, data-efficient TSFMs that combine interpretability, modularity, and predictive power.

Contributions We investigate the plausibility of LAFNs as candidate architectures for TSFMs, and provide preliminary evidence that these models can produce forecasts that generalize across domains, while capturing temporal and cross-feature data dependencies. Our contributions are as follows:

- We propose Likelihood-Aligned Forecast Networks (LAFN), a hypernetwork architecture designed for time-series forecasting, and demonstrate on-par or superior performance of these models as compared to existing TSFMs and widely-used statistical forecasting models.
- We propose a new loss function (Equation 1) for LAFN, which can in general be used to train neural networks that simultaneously predict point and probabilistic forecasts, which trade offs accuracy between the two during training.
- We introduce *CholeskySynth* (Algorithm 1), a new synthetic multivariate time-series data generation method that efficiently generalizes the KernelSynth method [7] from univariate to multivariate domains.

2 Likelihood-Aligned Forecast Networks

We refer the reader to Section A for the notational conventions we adopt, as well as for additional mathematical preliminaries and definitions of our evaluation metrics. We use $l, h, m \in \mathbb{N}$ as the *(historical) context length*, *forecast horizon*, and the number of variates, respectively.

Problem Formulation We assume time-series data $\mathcal{D} \doteq \{(l^{(k)}, h^{(k)}, m^{(k)}, \boldsymbol{Y}^{(k)}, \boldsymbol{Y}^{*(k)})\}_{k\in[n]}$, comprising $n \in \mathbb{N}$ multivariate samples s.t. for all $k \in [n]$, $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{m^{(k)}})^T \in \mathcal{Y}^{l^{(k)}\times m^{(k)}} \subseteq \mathbb{R}^{l^{(k)}\times m^{(k)}}$ is the *context* and $\boldsymbol{Y}^* = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{m^{(k)}})^T \in \mathcal{Y}^{h^{(k)}\times m^{(k)}} \subseteq \mathbb{R}^{l^{(k)}\times m^{(k)}}$ is the *target*. Our goal is to learn an *(uncertainty-aware) forecaster* $\boldsymbol{Y} \mapsto (\boldsymbol{F}^{\boldsymbol{\theta}}(\boldsymbol{Y}), \mu_{\boldsymbol{\theta}}(\boldsymbol{Y})) \in \mathcal{Y}^{h\times m} \times \Delta(\mathcal{Y}^{h\times m})$, which takes as input context $\boldsymbol{Y} \in \mathcal{Y}^{l\times m}$ of length $l \in \mathbb{N}$ and outputs *point forecasts* $\boldsymbol{F}^{\boldsymbol{\theta}}(\boldsymbol{Y})$ and *probability forecasts* $\mu_{\boldsymbol{\theta}}(\boldsymbol{Y})$ of future values of the $m \in \mathbb{N}$ variates for the next $h \in \mathbb{N}$ time-steps.[2]

While prior work on deterministic (resp. stochastic) forecasting typically formulates the learning problem as minimizing Euclidean error (resp. negative log-likelihood), there is no widely accepted learning objective for uncertainty-aware forecasters that jointly models both accuracy and uncertainty. A naive approach would be to minimize the arithmetic mean of Euclidean error and negative log-likelihood. However, such an objective is known to weight the two losses unevenly, often over-optimizing one at the expense of the other [8]. To address this imbalance, we propose minimizing their geometric mean, which corresponds to minimizing the mean of both objectives up to diminishing marginal returns in either loss component:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{Y},\boldsymbol{Y}^*)\sim\mathbb{P}(\mathcal{D})} \left[\exp\left\{\log\left\|\boldsymbol{Y}^* - \boldsymbol{F}^{\boldsymbol{\theta}}(\boldsymbol{Y})\right\|^2 + \log\left(\log\left(\frac{1}{\mu_{\boldsymbol{\theta}}(\boldsymbol{Y})}\right)\right)\right\}\right] \tag{1}$$

LAFNs name derives from the fact that they minimize this loss.

[2] A forecaster $\boldsymbol{F}^{\boldsymbol{\theta}}$ (resp. $\mu_{\boldsymbol{\theta}}$) alone is said to be a *deterministic* (resp. *stochastic*) forecaster.

Algorithm 1 CholeskySynth: Synthetic Multivariate Time-Series Generation

Input: Kernel banks $\mathcal{K}_{\boldsymbol{U}}, \mathcal{K}_{\boldsymbol{V}} \subseteq \{\mathbb{R} \times \mathbb{R} \to \mathbb{R}\}$, maximum kernels per axis $k_{\boldsymbol{U}}, k_{\boldsymbol{V}} \in \mathbb{N}$, time-series length p, number of variates m
Output: Synthetic multivariate time series $\boldsymbol{X} \in \mathbb{R}^{p \times m}$

$\kappa_{\boldsymbol{U}} \sim \text{Unif}\{1, \ldots, k_{\boldsymbol{U}}\}, \quad \kappa_{\boldsymbol{V}} \sim \text{Unif}\{1, \ldots, k_{\boldsymbol{V}}\}$ ▷ sample kernel counts
$\{g_1, \ldots, g_{\kappa_{\boldsymbol{U}}}\} \overset{\text{i.i.d.}}{\sim} \mathcal{K}_{\boldsymbol{U}}, \quad \{h_1, \ldots, h_{\kappa_{\boldsymbol{V}}}\} \overset{\text{i.i.d.}}{\sim} \mathcal{K}_{\boldsymbol{V}}$ ▷ draw kernels
$\boldsymbol{U} \leftarrow [g_1(t, t')]_{t,t' \in [p]}, \quad \boldsymbol{V} \leftarrow [h_1(t, t')]_{t,t' \in [m]}$ ▷ initialize Gram matrices
for $i = 2, \ldots, \kappa_{\boldsymbol{U}}$ **do**
    $\star_i \sim \{\text{PSD-preserving binary operators}\}$ ▷ e.g., +, ×
    $\boldsymbol{U} \leftarrow \boldsymbol{U} \ \star_i \ [g_i(t, t')]_{t,t' \in [p]}$ ▷ compose time kernels
for $i = 2, \ldots, \kappa_{\boldsymbol{V}}$ **do**
    $\star_k \sim \{\text{PSD-preserving binary operators}\}$ ▷ e.g., +, ×
    $\boldsymbol{V} \leftarrow \boldsymbol{V} \ \star_i \ [h_i(t, t')]_{t,t' \in [m]}$ ▷ compose time kernels
Compute lower Cholesky factor $\boldsymbol{A}$ s.t. $\boldsymbol{U} = \boldsymbol{A}\boldsymbol{A}^\top$
Compute upper Cholesky factor $\boldsymbol{B}$ s.t. $\boldsymbol{U} = \boldsymbol{B}^\top \boldsymbol{B}$.
$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{pm})$ ▷ sample from a matrix of standard normal random variables
Sample $\boldsymbol{Z} \leftarrow [z_{ij} \sim \mathcal{N}(0, 1)]_{i \in [p], j \in [m]}$
$\boldsymbol{X} \leftarrow \boldsymbol{A}\boldsymbol{Z}\boldsymbol{B}$ ▷ apply Cholesky factors to obtain matrix-normal sample
Return $\boldsymbol{X}$

Data Generation In order to minimize the learning objective defined in Equation (1), we need a suitable distribution $\mathbb{P}(\mathcal{D})$ for time-series data. Unfortunately, real-world multivariate time-series corpora are few and far between. To overcome this challenge, we introduce the *CholeskySynth* algorithm (Algorithm 1), a computationally efficient multivariate generalization of *KernelSynth* [7], which generates univariate time-series data. At a high level, our algorithm generates samples from a matrix-normal distribution [9], using two covariance matrices, $\boldsymbol{U}$ and $\boldsymbol{V}$, which are built using kernel functions, and intended to express temporal and cross-variate dependencies, respectively. *CholeskySynth* is able to generate multivariate time-series data by varying the generation procedure for the covariance matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively.

As there does not exist an open-source implementation of sampling from a matrix normal distribution, we generate these samples using Cholesky decompositions of the covariance matrices. We provide an efficient implementation of this routine in our code repo. In our implementation, a small diagonal jitter is added to the covariance matrices to ensure positive definiteness before Cholesky factorization, to ensure numerical stability. *CholeskySynth* scales cubically in the number of time steps and variables, due to the Cholesky decompositions, with quadratic computational memory requirements.

Forecaster Network Overview. With our data generation method in hand, the learning objective in Equation (1) can now be minimized via a stochastic gradient descent method. As such we we now turn to describing our forecaster, the Likelihood-Aligned Forecast Network (LAFN), a metanetwork [5, 6] for time-series data. LAFN $(\boldsymbol{H}, \boldsymbol{\pi}^{\boldsymbol{\phi}})$ comprises a *hypernetwork* [10–12] $(\boldsymbol{Y}; \boldsymbol{\theta}) \mapsto \boldsymbol{H}(\boldsymbol{Y}; \boldsymbol{\theta}) \doteq \boldsymbol{\phi}$, which takes as input the context $\boldsymbol{Y}$ and outputs weights $\boldsymbol{\phi}$, the latter of which parameterize a *target network* $(\boldsymbol{\tau}; \boldsymbol{\phi}) \mapsto \boldsymbol{\pi}(\boldsymbol{\tau}; \boldsymbol{\phi}) \doteq (\widehat{\boldsymbol{Y}}, \mu(\cdot))$, which takes as input a vector $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_h)$ of h future time steps to forecastand outputs point forecasts $\widehat{\boldsymbol{Y}}$ and a distribution of forecasts μ. We describe the inputs of the target network as a vector of time steps $\boldsymbol{\tau}$ rather than the forecast horizon h to emphasize the fact that the network reasons about the forecasts across all time steps $\boldsymbol{\tau}$ simultaneously, rather than autoregressively forecasting each individual time step one by one.

Hypernetwork Our hypernetwork comprises four patcher layers of different patch sizes (i.e., 8, 32, 64, 128), which embed the context along the variate dimension, process it into patches, and then embed along the time dimension. We used a context window of 2048, padding or trimming the inputs as appropriate for input contexts with a longer or shorter context window than the model's.

The output of each patch layer is passed into a context layer, which consists of 3 intrapatch layers that comprise gating layers with skip connections, feedforward layers with skip connections, and GELU activations that operate along the time and feature dimension. Finally, for each patch, the output of the intrapatch layer is projected into the parameter space of the target network.

Target network The target network is made up of three stacks of forecast layers, with each one comprising a gating layer with skip connections, a feedforward layer with skip connections, and GELU activations that operate on the input time steps $\boldsymbol{\tau}$. The output of the forecast layer is then fed into 1) a forecast head with a linear layer that outputs point forecasts; and 2) a distribution head with a feedforward linear layer that outputs mixture weights, as well as mean and standard deviation parameters that are used to define the stochastic forecast as a Gaussian mixture distribution. In our implementation, we choose $h \doteq 128$ as the length of $\boldsymbol{\tau}$; however, our model can forecast data of arbitrary length with appropriate trimming or autoregression.

Importantly, the target network does not use the position of future forecasts $\boldsymbol{\tau}$ to create embeddings, but rather uses $\boldsymbol{\tau}$ as direct inputs to be transformed by network. This small change enables the target network to predict seasonality and trend components of the data, which are a function of the time steps themselves rather than of the embedded time-series data provided by the hypernetwork. Further, as the hypernetwork's outputs depend on a transformation of the data based on various patch sizes, the weights it outputs for the target network allow the target network to accurately predict long-, medium-, and short-range trends and seasonalities in the data.

3 Results

We include a summary of the results of experiments with LAFN and other competing models in Table 1a. We generate our results using the TempusBench evaluation framework [13], We summarize all additional results in Section E.

Table 1: Average Win Rates for deterministic and probabilistic forecasting models.

(a) Average Win Rate for MAPE Metric.

Model Name	Average Win Rate
LAFN	0.7931
Timesfm	0.6730
Croston Classic	0.6164
Seasonal Naive	0.5849
Toto	0.5789
Varmax	0.5714
Arima	0.5346
Moment	0.5220
Lagllama	0.5031
Lstm	0.4969
Moirai	0.4966
Svr	0.4874
Tabpfn	0.4828
Random Forest	0.4748
Tiny Time Mixer	0.4151
Chronos	0.4025
Exponential Smoothing	0.3333
Prophet	0.3333
Theta	0.3300

(b) Average Win Rate for CRPS Metric

Model Name	Average Win Rate
Toto	1.0000
Moirai	0.7857
Lafn	0.5714
Chronos	0.4000
Lagllama	0.2667

4 Future Directions

We omit, for this workshop version of the LAFN paper, two ongoing research directions still under active development: (i) forecasting with covariates and (ii) the evaluation of probabilistic forecasting models. A more general and extended version of LAFN will be released in the coming months as part of the full paper.

References

[1] u/nkafr. The rise of foundation time-series forecasting models. https://www.reddit.com/r/datascience/comments/1e865bt/the_rise_of_foundation_timeseries_forecasting/, 2024. URL https://www.reddit.com/r/datascience/comments/1e865bt/the_rise_of_foundation_timeseries_forecasting/. Reddit post on r/datascience. 1

[2] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 1970. 1

[3] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995. 1

[4] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alexander Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997. 1

[5] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. doi: 10.1162/neco.1992.4.1.131. 2, 3

[6] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *International conference on machine learning*, pages 2554–2563. PMLR, 2017. 2, 3

[7] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024. 2, 3, 17, 30, 38

[8] Cyrus Cousins. An axiomatic theory of provably-fair welfare-centric machine learning. *Advances in Neural Information Processing Systems*, 34:16610–16621, 2021. 2

[9] Arjun K Gupta and Daya K Nagar. *Matrix variate distributions*. Chapman and Hall/CRC, 2018. 3

[10] David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=rkpACe1lx. 3

[11] Tomer Galanti and Lior Wolf. On the role of hypernetworks in generalization. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=rklp93EtvB.

[12] Szymon Klocek, Lukasz Maziarka, Michal Warchol, Jacek Tabor, Ryszard Nowak, and Tomasz Trzcinski. Hypernetwork functional image representation. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1c1ffda39a816b548bf9896b23b7af3c-Abstract.html. 3

[13] Denizalp Goktas, Amy Greenwald, Gerardo Riano-Briceno, Alexandra Magnusson, Alif Abdullah, and Beatriz de Lucio. Tempusbench: An evaluation framework for time-series forecasting. In *Recent Advances in Time Series Foundation Models Have We Reached the'BERT Moment'?* 4

[14] James Bradbury, Roy Frostig, Peter Hawkins, Matthew J. Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of python+numpy programs, 2018. URL http://github.com/google/jax. 12

[15] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. *arXiv preprint arXiv:2402.02592*, 2024. 13, 30

[16] Xu Liu, Juncheng Liu, Gerald Woo, Taha Aksu, Yuxuan Liang, Roger Zimmermann, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. Moirai-moe: Empowering time series foundation models with sparse mixture of experts. *arXiv preprint arXiv:2410.10469*, 2024. 16

[17] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning*, 2024. 16, 17, 30

[18] Shi Bin Hoo, Samuel Müller, David Salinas, and Frank Hutter. From tables to time: How tabpfn-v2 outperforms specialized time series forecasting models. *arXiv preprint arXiv:2501.02945*, 2025. 18

[19] Vijay Ekambaram, Arindam Jati, Pankaj Dayama, Sumanta Mukherjee, Nam Nguyen, Wesley M Gifford, Chandra Reddy, and Jayant Kalagnanam. Tiny time mixers (ttms): Fast pre-trained models for enhanced zero/few-shot forecasting of multivariate time series. *Advances in Neural Information Processing Systems*, 37:74147–74181, 2024. 18

[20] Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, et al. Lag-llama: Towards foundation models for probabilistic time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023. 19

[21] Ben Cohen, Emaad Khwaja, Youssef Doubli, Salahidine Lemaachi, Chris Lettieri, Charles Masson, Hugo Miccinilli, Elise Ramé, Qiqi Ren, Afshin Rostamizadeh, et al. This time is different: An observability perspective on time series foundation models. *arXiv preprint arXiv:2505.14766*, 2025. 19, 38

[22] Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. *arXiv preprint arXiv:2402.03885*, 2024. 19, 38

[23] Sima Siami-Namini and Akbar Siami Namin. Forecasting economics and financial time series: Arima vs. lstm. *arXiv preprint arXiv:1803.06386*, 2018. 20, 22

[24] Thomas R Willemain, Charles N Smart, Joseph H Shockor, and Philip A DeSautels. Forecasting intermittent demand in manufacturing: a comparative evaluation of croston's method. *International Journal of forecasting*, 10(4):529–538, 1994. 21

[25] Prajakta S Kalekar et al. Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi school of information Technology*, 4329008(13):1–13, 2004. 21

[26] Pooja Anand, Mayank Sharma, and Anil Saroliya. A comparative analysis of artificial neural networks in time series forecasting using arima vs prophet. In *2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE)*, pages 527–533. IEEE, 2024. 22

[27] Lin Lin, Fang Wang, Xiaolong Xie, and Shisheng Zhong. Random forests-based extreme learning machine ensemble for multi-regime time series prediction. *Expert Systems with Applications*, 83:164–176, 2017. 23

[28] Senyao Wang and Jin Ma. A novel ensemble model for load forecasting: Integrating random forest, xgboost, and seasonal naive methods. In *2023 2nd Asian Conference on Frontiers of Power and Energy (ACFPE)*, pages 114–118. IEEE, 2023. 23, 25

[29] Fan Zhang and Lauren J O'Donnell. Support vector regression. In *Machine learning*, pages 123–140. Elsevier, 2020. 24

[30] Dimitrios D Thomakos and Konstantinos Nikolopoulos. Forecasting multivariate time series with the theta method. *Journal of Forecasting*, 34(3):220–229, 2015. 24

[31] 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, pages 95–104,

New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356572. doi: 10.1145/3209978.3210006. URL https://doi.org/10.1145/3209978.3210006. 27

[32] Indeed. Software Development Job Postings on Indeed in the United States [IHLIDXUSTPSOFTDEVE]. https://fred.stlouisfed.org/series/IHLIDXUSTPSOFTDEVE, 2025. Retrieved August 29, 2025. 27

[33] Nisarg Chodavadiya. Daily Gold Price (2015-2021) Time Series. https://www.kaggle.com/datasets/nisargchodavadiya/daily-gold-price-20152021-time-series, 2025. Accessed on August 29, 2025. 27

[34] Coinbase. Coinbase Litecoin [CBLTCUSD]. https://fred.stlouisfed.org/series/CBLTCUSD, 2025. Retrieved August 29, 2025. 27

[35] IgnacioQG. 2001-2022 Hourly Dataset of Pollution in Madrid. https://www.kaggle.com/datasets/ignacioqg/20012022-hourly-dataset-of-pollution-in-madrid, 2022. Accessed on August 29, 2025. 27

[36] DeltaTrup. LT 1-Minute Historical Stock Data (2003-2024). https://www.kaggle.com/datasets/deltatrup/lt-1-minute-historical-stock-data-2003-2024, may 2024. Accessed on August 29, 2025. 27

[37] GabrielSantello. Airline Baggage Complaints – Time Series Dataset. https://www.kaggle.com/datasets/gabrielsantello/airline-baggage-complaints-time-series-dataset, 2023. Accessed on August 29, 2025. 27

[38] U.S. Census Bureau. Manufacturers: Inventories to Sales Ratio [MNFCTRIRSA]. https://fred.stlouisfed.org/series/MNFCTRIRSA, 2025. Retrieved August 29, 2025. 27

[39] Bank for International Settlements. Real Residential Property Prices for Germany. https://fred.stlouisfed.org/series/QDER628BIS, 2025. Retrieved August 29, 2025. 27

[40] Energy and Geoscience Institute at the University of Utah. Utah FORGE: Well 16A(78)-32 Drilling Data. Accessed via Data.gov, 2025. Accessed on August 29, 2025. 27

[41] Board of Governors of the Federal Reserve System (US). Federal Funds Effective Rate [FF]. https://fred.stlouisfed.org/series/FF, 2025. Retrieved August 29, 2025. 27

[42] U.S. Bureau of Economic Analysis. Personal Consumption Expenditures: Chain-type Price Index. https://fred.stlouisfed.org/series/DPCERG3A086NBEA, 2025. Retrieved August 29, 2025. 27

[43] SumanthVrao. Daily Climate Time Series Data. https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data, 2021. Accessed on August 29, 2025. 27

[44] BITS Pilani - Goa. SplitSmart: An Open Dataset for Enabling Research in Energy-Efficient Ductless-Split Air Conditioner, 2024. Accessed on August 29, 2025. 27

[45] City of New York. COVID-19 Daily Counts of Cases, Hospitalizations, and Deaths, 2025. Accessed on August 29, 2025. Daily count of NYC residents who tested positive for SARS-CoV-2, hospitalized with COVID-19, and deaths among COVID-19 patients. 27

[46] U.S. Bureau of Labor Statistics. All Employees, Health Care [CES6562000101]. https://fred.stlouisfed.org/series/CES6562000101, 2025. Retrieved August 29, 2025. 27

[47] NoeyIsLearning. Soil and Environmental Monitoring. https://www.kaggle.com/datasets/noeyislearning/soil-and-environmental-monitoring, 2024. Accessed on August 29, 2025. 27

[48] Riccardo Taormina et al. The Battle of the Attack Detection Algorithms: Disclosing Cyber Attacks on Water Distribution Networks. *Journal of Water Resources Planning and Management*, 144(8):04018048, aug 2018. doi: 10.1061/(ASCE)WR.1943-5452.0000969. URL https://www.batadal.net/data.html. 27

[49] Jorge Bañuelos-Gimeno, Natalia Sobrino, and Rosa Arce-Ruiz. Initial Insights into Teleworking's Effect on Air Quality in Madrid City. *Environments*, 11(9):204, 2024. doi: 10.3390/environments11090204. URL https://www.mdpi.com/2076-3298/11/9/204. 27

[50] RaminHuseyn. Web Traffic Time Series Dataset. https://www.kaggle.com/datasets/raminhuseyn/web-traffic-time-series-dataset, 2024. Accessed on August 29, 2025. 27

[51] UCI Machine Learning Repository. Hungarian Chickenpox Cases. https://doi.org/10.24432/C5103B, 2021. 27

[52] Alistair Johnson et al. MIMIC-III Clinical Database Demo (version 1.4). https://doi.org/10.13026/C2HM2Q, 2019. RRID:SCR_007345. 27

[53] Andrea Martiniano and Ricardo Ferreira. Absenteeism at work. https://doi.org/10.24432/C5X882, 2012. 27

[54] Daqing Chen. Online Retail. https://doi.org/10.24432/C5BW33, 2015. 27

[55] Paulo Cortez and Aníbal Morais. Forest Fires. https://doi.org/10.24432/C5D88D, 2007. 27

[56] Adarsh Pal Singh and Sachin Chaudhari. Room Occupancy Estimation. https://doi.org/10.24432/C5P605, 2018. 27

[57] Yuxuan Liang, Haomin Wen, Yuqi Nie, Yushan Jiang, Ming Jin, Dongjin Song, Shirui Pan, and Qingsong Wen. Foundation models for time series analysis: A tutorial and survey. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6555–6565, 2024. 30

[58] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 5 edition, 2015. 38

[59] Rob J. Hyndman, Anne B. Koehler, J. Keith Ord, and Ralph D. Snyder. *Forecasting with Exponential Smoothing: The State Space Approach*. Springer, 2008. 38

[60] Vassilis Assimakopoulos and Konstantinos Nikolopoulos. The theta model: A decomposition approach to forecasting. *International Journal of Forecasting*, 16(4):521–530, 2000. 38

[61] Helmut Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, 2005. 38

[62] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks, 2017. 38

[63] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations (ICLR)*, 2020. 38

[64] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023. 38

[65] Abhimanyu Das et al. Long-term forecasting with tide: Time-series dense encoder. In *Neural Information Processing Systems (NeurIPS)*, 2023. 38

[66] Bryan Lim, Sercan Ö. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2019. 38

[67] Yifan Nie, Zhihan Huang, Li Wang, Yuheng Sun, Yating He, and Zhifeng Zhang. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations (ICLR)*, 2023. 38

[68] Han Liu et al. itransformer: Inverted transformers are effective for time series forecasting, 2023. 38

[69] Kashif Rasul, V Ashkinazi, I Schuster, A Schneider, and A Mishkin. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *Neural Information Processing Systems (NeurIPS)*, 2021. 38

[70] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. In *Neural Information Processing Systems (NeurIPS)*, 2021. 38

[71] Kashif Rasul et al. Multivariate probabilistic time series forecasting via conditioned normalizing flows, 2020. 38

[72] Sangwoo Woo et al. Moirai: Foundation models for time series forecasting. In *International Conference on Learning Representations (ICLR)*, 2024. 38

[73] Abhimanyu Das et al. Timesfm: Time series foundation models at scale, 2023. 38

[74] Kashif Rasul et al. Lag-llama: Towards foundation models for time series forecasting, 2023. 38

[75] Yan Liu et al. Timer: Efficient time-series foundation model, 2024. 38

[76] Tian Gao et al. Units: Universal time series foundation models, 2024. 38

[77] Vinay Ekambaram et al. Multi-level tiny time mixers for efficient time-series foundation models, 2024. 38

[78] Xing Chen et al. Visionts: Multimodal time-series foundation models, 2024. 38

[79] Andreas Auer, Patrick Podest, Daniel Klotz, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. Tirex: Zero-shot forecasting across long and short horizons with enhanced in-context learning. *arXiv preprint arXiv:2505.23719*, 2025. 38

[80] Spyros Makridakis and Michele Hibon. The M3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000. 38

[81] Spyros Makridakis, Evangelos Spiliotis, and Vassilis Assimakopoulos. The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4): 802–808, 2018. 38

[82] Spyros Makridakis, Evangelos Spiliotis, and Vassilis Assimakopoulos. The M5 accuracy competition: Results, findings and conclusions. *International Journal of Forecasting*, 38(4): 1346–1364, 2022. 38

[83] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, et al. Monash time series forecasting archive. In *NeurIPS Datasets and Benchmarks Track*, 2021. 38

[84] Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S Jensen, Zhenli Sheng, et al. Tfb: Towards comprehensive and fair benchmarking of time series forecasting methods. *arXiv preprint arXiv:2403.20150*, 2024. 38

[85] Haixu Zhang et al. Probts: Benchmarking probabilistic forecasting. In *Neural Information Processing Systems (NeurIPS)*, 2023. 38

[86] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1): 37–45, 2018. 38

[87] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, et al. sktime: A unified interface for machine learning with time series. In *Proceedings of the 2nd Workshop on Systems for ML (NeurIPS)*, 2019. 38

[88] Alexander Alexandrov et al. Gluonts: Probabilistic time series models in python, 2020. 38

[89] Kashif Rasul. Pytorchts: A probabilistic deep learning library for time series, 2021. GitHub repository: https://github.com/zalandoresearch/pytorch-ts. 38

[90] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. 38

[91] Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. 38

A Additional Mathematical Background

A.1 Mathematical notation

We adopt the following calligraphic conventions to insist on the nature of the mathematical object at hand: We use calligraphic uppercase letters to denote sets (e.g., $\mathcal{X}$), bold uppercase letters to denote matrices (e.g., $\boldsymbol{X}$), bold lowercase letters to denote vectors (e.g., $\boldsymbol{p}$), lowercase letters to denote scalar quantities (e.g., x), and uppercase letters to denote random variables (e.g., X). We denote the ith row vector of a matrix (e.g., $\boldsymbol{X}$) by the corresponding bold lowercase letter with subscript i (e.g., $\boldsymbol{x}_i$). Similarly, we denote the jth entry of a vector (e.g., $\boldsymbol{p}$ or $\boldsymbol{x}_i$) by the corresponding lowercase letter with subscript j (e.g., p_j or x_{ij}). We denote functions by a letter determined by the value of the function, e.g., f if the mapping is scalar valued, $\boldsymbol{f}$ if the mapping is vector valued, and $\mathcal{F}$ if the mapping is set valued.

We denote the set $\{1, \ldots, n\}$ by $[n]$, the set $\{n, n+1, \ldots, m\}$ by $[n : m]$, the set of natural numbers by $\mathbb{N}$, and the set of real numbers by $\mathbb{R}$. We denote the positive and strictly positive elements of a set using a $+$ or $++$ subscript, respectively, e.g., $\mathbb{R}_+$ and $\mathbb{R}_{++}$. For any $n \in \mathbb{N}$, we denote the n-dimensional vector of zeros and ones by $\mathbf{0}_n$ and $\mathbf{1}_n$, respectively.

A.2 Mathematical Definitions

We let $\Delta_n = \{\boldsymbol{x} \in \mathbb{R}^n_+ \mid \sum_{i=1}^n x_i = 1\}$ denote the unit simplex in $\mathbb{R}^n$, and $\Delta(A)$ denote the set of all probability measures over a given set A. We also define the support of a probability density function $f \in \Delta(\mathcal{X})$ as $\operatorname{supp}(f) \doteq \{\boldsymbol{x} \in \mathcal{X} \mid f(\boldsymbol{x}) > 0\}$. Finally, we denote the orthogonal projection operator onto a set C by Π_C, i.e., $\Pi_C(\boldsymbol{x}) \doteq \arg\min_{\boldsymbol{y} \in C} \|\boldsymbol{x} - \boldsymbol{y}\|^2$.

A.3 Evaluation Metrics

An evaluation metric $\ell : \mathcal{Y}^h \times \mathcal{Y}^h \to \mathbb{R}_+$ is a positive-, scalar-valued function s.t. for any forecast $\widehat{\boldsymbol{Y}} \in \mathcal{Y}^h$ and realized future target values $\boldsymbol{Y}^* \in \mathcal{Y}^h$, $\ell(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}^*) \geq 0$ denotes the distance between the forecast and the realized values. We consider the following evaluation metrics at present. The *mean absolute error (MAE)* is defined as $\ell^{\mathrm{MAE}}(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}^*) \doteq \frac{1}{mh} \sum_{i \in [m]} \sum_{\boldsymbol{\tau}=1}^{h} |\widehat{y}_{i\boldsymbol{\tau}} - y^*_{i\boldsymbol{\tau}}|$. The *mean squared error (MSE)* is defined as $\ell^{\mathrm{MSE}}(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}^*) \doteq \frac{1}{mh} \sum_{i \in [m]} \sum_{\boldsymbol{\tau}=1}^{h} (\widehat{y}_{i\boldsymbol{\tau}} - y^*_{i\boldsymbol{\tau}})^2$. The *mean absolute scale error (MASE)* is defined as $\ell^{\mathrm{MASE}}(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}^*) \doteq \frac{1}{mh} \sum_{i \in [m]} \sum_{\boldsymbol{\tau}=1}^{h} \frac{|\widehat{y}_{i\boldsymbol{\tau}} - y^*_{i\boldsymbol{\tau}}|}{\frac{1}{h-1} \sum_{\boldsymbol{\tau}=1}^{l-1} |y_{i\boldsymbol{\tau}+1} - y_{i\boldsymbol{\tau}}|}$.[3]

[3]We note MAE is scale-dependent but less sensitive to outliers, MSE disproportionately penalizes large forecast errors and is therefore more outlier-sensitive, while MASE normalizes errors w.r.t. the forecasts of naive forecast method (i.e., setting the next time-step's forecast to be the current time-step realized value), making it scale-free and comparable across datasets or domains.

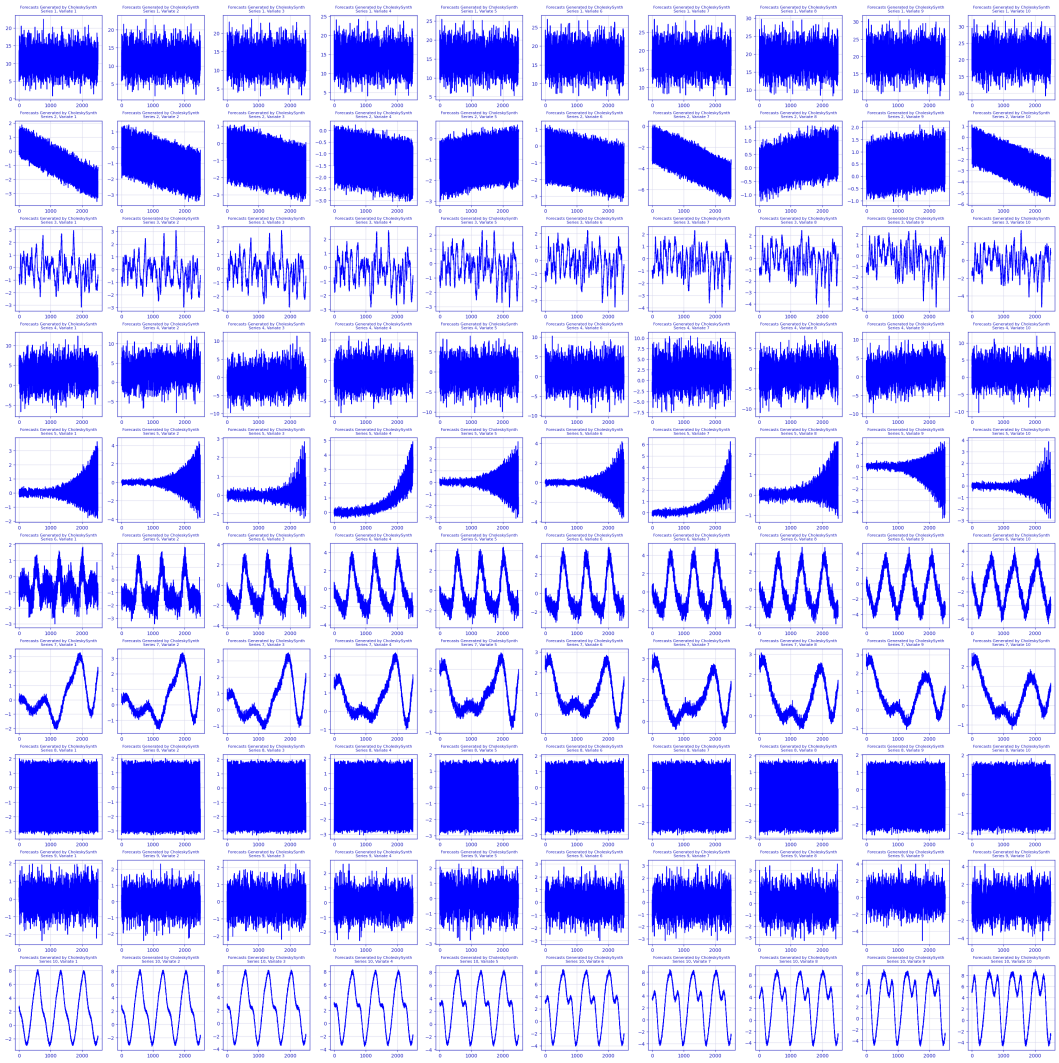
B Additional Details on CholeskySynth

We provide in our code repo (https://github.com/Smlcrm/CholeskySynth) an efficient implementation of the CholeskySynth method (**??**) in Jax [14]. We summarize the kernels used to generate the data used in the training of LAFN in Table 2, and note that we have used $\kappa_{\boldsymbol{U}} \doteq 4$ and $\kappa_{\boldsymbol{V}} \doteq 3$. We include in Figure 1 visualizations of multivariates time-series generated by CholeskySynth under the aforementioned parameter choices.

Table 2: Kernel bank used for data generation.

Kernel	**Formula**	**Hyperparameters**
Constant	$\kappa_{\text{Const}}(x, x') = C$	$C = 1$
White Noise	$\kappa_{\text{White}}(x, x') = \sigma_n^2 \, \mathbf{1}_{x=x'}$	$\sigma_n^2 \in \{0.1, 1.0\}$
Polynomial	$\kappa_{\text{Poly}}(x, x') = (\sigma^2 + x^\top x')^\alpha$	$\sigma \in \{0, 1, 10\},\ \alpha \in \{1, 2, 3\}$
RBF	$\kappa_{\text{RBF}}(x, x') = \exp\Big(-\frac{\|x - x'\|^2}{2\ell^2}\Big)$	$\ell \in \{0.01, 0.1, 1, 10\}$
Rational Quadratic	$\kappa_{\text{RQ}}(x, x') = \Big(1 + \frac{\|x - x'\|^2}{2\alpha\ell^2}\Big)^{-\alpha}$	$\alpha \in \{0.01, 0.1, 1, 10\},\ \ell = 1$
Periodic	$\kappa_{\text{Per}}(x, x') = \exp\Big(-2\sin^2(\pi\frac{\|x - x'\|}{p})\Big)$	$p \in \{\frac{24}{2048}, \frac{48}{2048}, \frac{96}{2048}, \frac{24\cdot7}{2048}, \frac{48\cdot7}{2048}, \frac{96\cdot7}{2048}, \frac{7}{2048}, \frac{14}{2048}, \frac{30}{2048}, \frac{60}{2048}, \frac{365}{2048}, \frac{730}{2048}, \frac{4}{2048}, \frac{26}{2048}, \frac{52}{2048}, \frac{6}{2048}, \frac{12}{2048}, \frac{40}{2048}, \frac{10}{2048}\}$

Figure 1: Examples of synthetic time series generated via Cholesky-based sampling. Each row is a sample from CholeskySynth, and each column is a variate. Each subplot represents a different covariance structure or noise process.



C Forecasting Models Included in LAFN's evaluation

In this section, we summarize the forecasting models which have been included in the evaluation LAFN. We summarize all models in Table 3, and provide and comparison of TSFMs, machine learning forecasting models, and statistical forecasting models in Table 4.

C.0.1 Moirai

Moirai is a universal time series forecasting model developed by Salesforce AI Research, built upon a masked encoder-only Transformer architecture. It is designed as a single, large pre-trained model capable of handling diverse forecasting tasks without dataset-specific retraining. The model is pre-trained on LOTSA, a large-scale archive of over 27 billion observations, enabling it to perform powerful zero-shot forecasting. [15]

- **Input:** Accepts univariate or multivariate time series with an arbitrary number of variates and covariates.

Table 3: Summary of forecasters included in TempusBench.

Category	Included Models	Core Characteristics
Foundation Models	*Moirai, Moirai-MoE, TimesFM, TimesFM-2.0, Chronos, Lag-Llama, Toto, MOMENT, TTM, TabPFN-TS*	**Paradigm:** Universal, zero-shot/few-shot forecasting. A single large model is pre-trained on massive, diverse datasets and generalizes to new tasks without retraining. **Architecture:** Primarily based on Transformers or other deep learning structures like MLP-Mixers. They process raw time series via patching or novel tokenization schemes. **I/O:** Often produce probabilistic forecasts and can natively handle univariate, multivariate, and covariate data.
Classic Machine Learning	*LSTM, Random Forest, XGBoost, SVR*	**Paradigm:** Supervised learning models trained per-dataset. They excel at capturing complex, nonlinear relationships but require specific training for each task. **Architecture:** Diverse, including Recurrent Neural Networks (for sequence memory), Tree Ensembles (for interaction effects), and Kernel Methods. **I/O:** Typically require explicit feature engineering (e.g., lags, calendar variables) to create a tabular format. Most often produce point forecasts.
Statistical & Decomposable	*ARIMA, Holt-Winters, Prophet, Theta Method, Croston's Method, Seasonal Naive*	**Paradigm:** Assume the time series is generated by an underlying statistical process or can be decomposed into simpler, interpretable components like trend and seasonality. **Architecture:** An explicit mathematical formula is fitted directly to an individual time series. **I/O:** Highly interpretable point forecasts. Often specialized for particular data patterns (e.g., intermittency with *Croston's*).

Table 4: Comparative Overview of Forecasting Models

Feature	Group 1: Foundation Models	Group 2: Machine Learning Models	Group 3: Classical & Statistical Models
Models Included	• *Moirai / Moirai-MoE* • *TimesFM / TimesFM-2.0* • *Chronos* • *Lag-Llama* • *Toto* • *MOMENT* • *TabPFN-TS* • *Tiny Time Mixers (TTM)*	• *LSTM* • *Random Forest* • *XGBoost* • *SVR*	• *ARIMA* • *Holt-Winters* • *Prophet* • *Theta Method* • *Croston's Method* • *Seasonal Naive*
Core Paradigm	Large, pre-trained models designed for universal, zero-shot forecasting. They learn general time-series patterns from massive, diverse datasets.	Models trained for a specific forecasting task, often relying on feature engineering. They leverage distinct architectures (e.g., recurrence, ensembles) rather than massive pre-training.	Model-based approaches assuming an underlying stochastic process or decomposable structure. Parameters are estimated directly from the target time series.
Architecture	Primarily Transformer-based (Encoder, Decoder, or both). Innovations include MoE layers, residual forecasting, and specialized attention mechanisms.	Diverse architectures: MLP-Mixer (*TTM*), RNN (*LSTM*), Tabular-Transformer (*TabPFN-TS*), Tree Ensembles (*RF, XGBoost*), and Kernel-based (*SVR*).	Mathematical formulations: State-space models (*ARIMA, Holt-Winters*), decomposable additive models (*Prophet, Theta*), and simple heuristics (*Croston's, S. Naive*).
Input Handling	Process raw time series, typically via patching (*Moirai, TimesFM*), lag-based tokenization (*Lag-Llama*), or value quantization (*Chronos*). Can natively handle uni/multivariate series.	Generally require explicit feature engineering (e.g., lags, calendar variables) to create a tabular dataset (*RF, XGBoost, SVR*). *LSTM* and *TTM* process raw sequences.	Operate directly on the univariate time series. May require stationarity (*ARIMA*) or be specialized for patterns like seasonality (*Holt-Winters*) or intermittency (*Croston's*).
Output Type	Mostly probabilistic, predicting the parameters of a flexible distribution. Point forecasts are derived from the distribution (e.g., median).	Primarily point forecasts. *TabPFN-TS* is a notable exception, providing a probabilistic output by approximating the posterior.	Primarily point forecasts. *Prophet* is an exception, generating uncertainty intervals. Probabilistic versions exist but are not standard.
Key Trait	Powerful zero-shot/few-shot performance. High model capacity and ability to generalize across domains without dataset-specific training.	Model-specific strengths: computational efficiency (*TTM*), modeling long-term dependencies (*LSTM*), and capturing complex non-linear interactions (*RF, XGBoost*).	High interpretability and strong statistical foundations. Often specialized and highly efficient for specific data patterns (e.g., trend, seasonality, intermittency).

- **Output:** Produces a probabilistic forecast by predicting the parameters of a flexible mixture distribution (composed of Student's t, Negative Binomial, Log-Normal, and low-variance Normal distributions).
- **Architecture:** Employs a masked encoder-only Transformer. Its key innovations include:
  - **Multi Patch Size Projection:** Uses different patch sizes to effectively process time series of varying frequencies.
  - **Any-variate Attention:** Flattens multivariate series into a single sequence and uses binary attention biases to manage an arbitrary number of variates while maintaining permutation equivariance.
- **Forecasting Type:** A universal, zero-shot, probabilistic forecaster. It can generate point forecasts by taking the median of the predicted distribution.

C.0.2 Moirai-MoE

Moirai-MoE is an advanced version of the Moirai foundation model that integrates a Sparse Mixture of Experts (MoE) architecture. Instead of relying on heuristic-based, frequency-specific projection layers, Moirai-MoE delegates the task of modeling diverse time series patterns to specialized "expert" networks within its Transformer layers. This allows for automatic, token-level specialization in a data-driven manner, leading to improved accuracy and greater efficiency in terms of activated parameters. [16]

- **Input:** Accepts univariate or multivariate time series with an arbitrary number of variates and covariates.
- **Output:** Produces a probabilistic forecast by predicting the parameters of a flexible mixture distribution for the next token in an autoregressive manner.
- **Architecture:** Employs a decoder-only Transformer that replaces the standard Feed-Forward Network (FFN) layers with MoE layers. Key architectural changes from the original Moirai include:
  - **Mixture of Experts (MoE):** A gating function routes each time series token to a small subset of specialized expert networks, allowing the model to handle diverse patterns at a granular level.
  - **Single Projection Layer:** It uses a single input/output projection layer for all time series, removing the dependency on frequency-based heuristics.
- **Forecasting Type:** A universal, zero-shot, probabilistic forecaster that is more accurate and efficient (in terms of activated parameters) than the original Moirai model. It can generate point forecasts by taking the median of the predicted distribution.

C.0.3 TimesFM

TimesFM is a time-series foundation model developed by Google Research, designed for zero-shot forecasting. It is based on a decoder-only Transformer architecture and is pretrained on a very large corpus of time series data, combining both real-world and synthetic sources. The model's key objective is to provide accurate out-of-the-box point forecasts on unseen datasets without requiring any dataset-specific training. [17]

- **Input:** Accepts a univariate time series context window.
- **Output:** Produces a point forecast for a given prediction horizon.
- **Architecture:** Employs a decoder-only Transformer architecture that processes the time series in patches. Key architectural features include:
  - **Decoder-Only Transformer:** Utilizes a standard decoder-style attention mechanism to autoregressively predict future values patch by patch.
  - **Input Patching:** The input time series is segmented into non-overlapping patches, which are then embedded using a residual block of MLPs before being fed to the Transformer.
- **Forecasting Type:** A universal, zero-shot, point forecaster designed primarily for long-horizon forecasting tasks.

C.0.4 TimesFM-2.0

TimesFM-2.0 is an improved version of the original foundation model from Google Research. While retaining the same decoder-only Transformer architecture, its key innovation lies in forecasting the residual component of a time series after performing a seasonal-trend decomposition. This approach makes the model significantly more accurate, particularly for time series that exhibit clear trends. [17]

- **Input:** Accepts a univariate time series context window.
- **Output:** Produces a point forecast for a given prediction horizon.
- **Architecture:** Based on the original decoder-only Transformer with input patching. The primary architectural update is its **residual forecasting** methodology:
  - **Seasonal-Trend Decomposition:** The model first decomposes the input series to separate its trend and seasonal components.
  - **Residual Forecasting:** The core Transformer then forecasts the residual (the signal remaining after decomposition). This forecast is added back to the projected trend to produce the final prediction.
- **Forecasting Type:** A universal, zero-shot, point forecaster with enhanced performance on trended time series compared to its predecessor.

C.0.5 Chronos

Chronos is a family of pretrained time series models developed by Amazon Science that frames forecasting as a language modeling task. The core idea is to "tokenize" time series values by scaling and quantizing them into a fixed vocabulary. By doing so, standard Transformer-based language model architectures can be trained on sequences of these tokens using a cross-entropy loss, effectively learning the "language" of time series. [7]

- **Input:** Accepts a univariate time series context window.
- **Output:** Produces a probabilistic forecast by generating multiple sample future trajectories. A point forecast can be derived from the median of these samples.
- **Architecture:** Based on standard language model architectures (specifically the T5 encoder-decoder family). Its defining characteristic is its unique data preprocessing pipeline:
  - **Tokenization via Quantization:** The model first applies mean scaling to the input time series. It then quantizes these scaled values into a finite set of discrete tokens, converting the continuous series into a sequence of categorical variables.
  - **Language Model Training:** The model is trained to predict the next token in a sequence using a standard cross-entropy loss, analogous to how a language model predicts the next word.
- **Forecasting Type:** A universal, zero-shot, probabilistic forecaster.

C.0.6 TabPFN

TabFPN is a forecasting framework that adapts feature pyramid networks (FPN), originally developed for computer vision tasks, to tabular time-series data. The approach builds hierarchical feature representations across multiple temporal resolutions, enabling the model to capture both short- and long-range dependencies. Unlike traditional time-series architectures, TabFPN treats forecasting as a structured feature-learning problem on tabularized sequences, combining multiscale decomposition with probabilistic prediction.

- **Input:** A univariate or multivariate time series, converted into tabular form with hierarchical features at multiple temporal resolutions.
- **Output:** Produces probabilistic forecasts by estimating distributions over future values at each horizon; point forecasts can be obtained from the distribution mean or median.
- **Architecture:**

  - *Feature Pyramids:* The series is decomposed into multiple temporal scales (e.g., short-term, medium-term, seasonal) using windowed transformations. Each scale yields a feature representation.
  - *FPN Backbone:* These features are passed into a feature pyramid network adapted for tabular regression, allowing cross-scale information flow and refinement.
  - *Prediction Head:* Aggregates multiscale features to generate forecasts, with uncertainty quantification via distributional outputs.
- **Forecasting Type:** A universal, zero-shot, probabilistic forecaster with explicit multiscale feature integration.

C.0.7 TabPFN-TS

TabPFN-TS is a novel approach that adapts TabPFN-v2, a general-purpose tabular foundation model, for time series forecasting. The core methodology involves recasting the forecasting problem as a tabular regression task. This is achieved through lightweight feature engineering on the time index, without relying on lagged values. Notably, the underlying TabPFN-v2 model was pretrained exclusively on synthetic tabular data and has not seen any time series data. [18]

- **Input:** A univariate time series, which is converted into a feature matrix based on timestamps.
- **Output:** Produces a probabilistic forecast by approximating the posterior predictive distribution for each future time step. Point forecasts can be derived from the mean or median of this distribution.
- **Architecture:** It does not use a time-series-specific architecture. Instead, it relies on:
  - **Feature Engineering:** The time series is transformed into a tabular dataset by creating features from timestamps. These include standard calendar features (e.g., hour of day, day of week), automatically detected seasonal features via a Fourier transform, and a simple running index.
  - **TabPFN-v2 Model:** The generated tabular data is fed into the pretrained TabPFN-v2 model, which performs the regression task to predict future values.
- **Forecasting Type:** A universal, zero-shot, probabilistic forecaster.

C.0.8 Tiny Time Mixers (TTM)

Tiny Time Mixers (TTM) is a family of lightweight pre-trained models from IBM Research, based on the efficient TSMixer architecture. In contrast to large, LLM-based approaches, TTMs are designed to be extremely small (<1M parameters) and fast, while still providing strong zero-shot and few-shot forecasting performance. The models are pre-trained exclusively on a large corpus of public time series datasets, making them a highly efficient alternative for universal forecasting. [19]

- **Input:** Accepts univariate or multivariate time series, with optional support for exogenous variables during the fine-tuning stage.
- **Output:** Produces a point forecast for a given prediction horizon.
- **Architecture:** Based on the MLP-Mixer architecture. The model is pre-trained in a channel-independent manner and uses a multi-level structure to handle diverse data and tasks.
  - **TSMixer Backbone:** The core of the model uses simple MLP blocks for temporal and feature mixing, avoiding the computational overhead of Transformer-based attention.
  - **Multi-Resolution Pre-training:** Employs several novel techniques to handle heterogeneous datasets, including **adaptive patching** (using different patch configurations at different layers) and **data augmentation via downsampling**.
  - **Multi-level Modeling:** Uses a frozen pre-trained backbone and a smaller, fine-tunable decoder, which can incorporate channel-mixing and an **exogenous mixer** to fuse external signals for target-specific tasks.
- **Forecasting Type:** A universal, zero-shot/few-shot, point forecaster, notable for its small size and computational efficiency.

C.0.9 Lag-Llama

Lag-Llama is a foundation model for univariate probabilistic time series forecasting. It is built upon a decoder-only Transformer architecture, similar to LLaMA, and is pretrained on a large, diverse corpus of open-source time series data. The model's key innovation is its tokenization strategy, which uses lagged values of the time series as input features, allowing it to generalize across different frequencies and domains. [20]

- **Input:** Accepts a univariate time series context window.
- **Output:** Produces a probabilistic forecast by outputting the parameters of a Student's t-distribution for the next time step. Future trajectories are generated autoregressively.
- **Architecture:** Based on a decoder-only Transformer (LLaMA). Its defining characteristic is its input representation:
  - **Tokenization via Lag Features:** Instead of patching, the input token for each time step is a vector composed of lagged values from the time series history (e.g., values from 1, 7, and 14 days prior). This is augmented with standard date-time features.
  - **Value Scaling:** Applies robust scaling (using median and IQR) to normalize the input values and includes the scaling parameters as additional features.
- **Forecasting Type:** A universal, zero-shot/few-shot, probabilistic forecaster.

C.0.10 Toto

Toto (Time Series Optimized Transformer for Observability) is a foundation model from Datadog, specifically designed for multivariate time series forecasting with a focus on observability metrics. It is built on a decoder-only Transformer architecture and incorporates several novel components to handle the unique challenges of observability data, such as high non-stationarity and heavy-tailed distributions. The model is pretrained on a large and diverse corpus that includes real-world observability data, public datasets, and synthetic data. [21]

- **Input:** Accepts multivariate time series.
- **Output:** Produces a probabilistic forecast by predicting the parameters of a Student-T mixture model.
- **Architecture:** A decoder-only Transformer with several key innovations tailored for observability data:
  - **Patch-based Causal Normalization:** A novel per-patch scaling method that computes normalization statistics from current and past data to handle highly nonstationary series.
  - **Proportional Factorized Attention:** An efficient attention mechanism that uses a mix of time-wise and variate-wise attention blocks to judiciously model interactions in high-dimensional multivariate data.
  - **Student-T Mixture Model Head:** An output layer that models the predictive distribution as a mixture of Student-T distributions to better capture the complex, heavy-tailed nature of observability metrics.
  - **Composite Robust Loss:** A hybrid loss function combining negative log-likelihood with a robust point-wise loss to stabilize training in the presence of outliers.
- **Forecasting Type:** A universal, zero-shot, probabilistic forecaster for multivariate time series.

C.0.11 MOMENT

MOMENT (Multi-task, Open-source, Foundation Model for Time-series) is a family of open-source foundation models from Carnegie Mellon University designed for general-purpose time series analysis. The models are built on a Transformer encoder architecture and are pretrained on a large, diverse collection of public time series called the "Time Series Pile." A key characteristic of MOMENT is its versatility; it is designed to serve as a building block for a wide range of downstream tasks, including forecasting, classification, anomaly detection, and imputation, often with minimal task-specific fine-tuning. [22]

- **Input:** Accepts a univariate time series of a fixed length. Multivariate time series are handled by treating each channel independently.
- **Output:** Produces a reconstructed version of the input time series. This output can be adapted for various downstream tasks, such as generating forecasts by masking future values or extracting embeddings for classification.
- **Architecture:** A standard Transformer encoder that processes time series data in patches.
  - **Masked Pre-training:** The model is pretrained using a masked time series prediction task. It learns to reconstruct randomly masked patches of the input time series, enabling it to learn robust representations.
  - **Patching:** The input time series is segmented into non-overlapping patches, which are then linearly projected into embeddings for the Transformer.
  - **Lightweight Prediction Head:** A simple linear layer is used to reconstruct the time series from the Transformer's output embeddings. This head can be easily replaced or adapted for different downstream tasks.
- **Forecasting Type:** A universal foundation model for general time series analysis. It can be used for zero-shot or few-shot forecasting (point-based), classification, anomaly detection, and imputation.

C.0.12 ARIMA

The Autoregressive Integrated Moving Average (ARIMA) model is a class of statistical models for analyzing and forecasting time series data. It is a generalization of the simpler Autoregressive Moving Average (ARMA) model that can be applied to non-stationary time series. The model's name reflects its three core components: Autoregression (AR), Integrated (I), and Moving Average (MA). These components capture the key temporal structures within the data, such as dependencies on past observations and past forecast errors. [23]

- **Input:** A univariate time series.
- **Output:** A point forecast for future time steps. While classical ARIMA produces point forecasts, probabilistic forecasts can be generated by assuming a distribution for the error term.
- **Mathematical Formulation:** An ARIMA(p, d, q) model is defined by three parameters: the order of the autoregressive component (p), the degree of differencing (d), and the order of the moving average component (q). The model assumes that the differenced time series, $\widetilde{\boldsymbol{y}}_t = (1-B)^d y_t$, is stationary, where B is the backshift operator. The formulation for the stationary series $\widetilde{\boldsymbol{y}}_t$ is:

$$\widetilde{\boldsymbol{y}}_t = c + \sum_{i=1}^{p} \phi_i \widetilde{\boldsymbol{y}}_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t \tag{2}$$

  where:
  - p is the autoregressive order, representing the number of lagged observations included in the model.
  - d is the degree of differencing, representing the number of times the raw observations are differenced to achieve stationarity.
  - q is the moving average order, representing the size of the moving average window applied to past forecast errors.
  - $\boldsymbol{\phi}$ is the vector of autoregressive coefficients.
  - $\boldsymbol{\theta}$ is the vector of moving average coefficients.
  - c is a constant term.
  - ϵ_t is the white noise error term at time t, typically assumed to be drawn from a Gaussian distribution with zero mean.
- **Forecasting Type:** A statistical model that provides point forecasts. It is often used as a baseline in forecasting tasks. Seasonal variations can be included by using a Seasonal ARIMA (SARIMA) model.

C.0.13 Croston's Method

Croston's method is a forecasting technique specifically designed for intermittent demand time series, which are characterized by sporadic, non-zero values interspersed with periods of zero demand. The method decomposes the original time series into two separate components: the magnitude of the non-zero demand and the time interval between consecutive demands. By forecasting these two components separately using Simple Exponential Smoothing and then combining them, the model provides a more accurate estimate of the mean demand per period compared to standard smoothing methods, which can be biased when applied to intermittent data. [24]

- **Input:** A univariate time series with intermittent demand.
- **Output:** A point forecast for the average demand per period.
- **Mathematical Formulation:** The method maintains and updates two estimates: one for the non-zero demand size ($\widehat{\boldsymbol{z}}$) and one for the interval between demands ($\widehat{\boldsymbol{p}}$). Let y_t be the demand at time t, and let q be the time elapsed since the last demand. The updates occur only when a non-zero demand is observed ($y_t > 0$):

$$\widehat{\boldsymbol{z}}_t = \widehat{\boldsymbol{z}}_{t-1} + \alpha(y_t - \widehat{\boldsymbol{z}}_{t-1}) \tag{3}$$

$$\widehat{\boldsymbol{p}}_t = \widehat{\boldsymbol{p}}_{t-1} + \alpha(q - \widehat{\boldsymbol{p}}_{t-1}) \tag{4}$$

  If demand at time t is zero, the estimates are not updated ($\widehat{\boldsymbol{z}}_t = \widehat{\boldsymbol{z}}_{t-1}$, $\widehat{\boldsymbol{p}}_t = \widehat{\boldsymbol{p}}_{t-1}$) and the interval counter q is incremented. After a demand occurs, q is reset to 1. The final forecast for the mean demand per period, $\widehat{\boldsymbol{y}}_t$, is the ratio of the two smoothed components:

$$\widehat{\boldsymbol{y}}_t = \frac{\widehat{\boldsymbol{z}}_t}{\widehat{\boldsymbol{p}}_t} \tag{5}$$

  where α is the smoothing parameter.
- **Forecasting Type:** A statistical model for point forecasting, specialized for intermittent or "lumpy" demand patterns.

C.0.14 Holt-Winters Exponential Smoothing

Holt-Winters is an extension of exponential smoothing that explicitly models trend and seasonality. It is a widely used statistical method for forecasting time series data that exhibit these components. The method operates by applying exponential smoothing to three components: the level, the trend, and the seasonality. There are two primary variations of the model, additive and multiplicative, which differ in how they incorporate the seasonal component. [25]

- **Input:** A univariate time series with trend and seasonality.
- **Output:** A point forecast for future time steps.
- **Mathematical Formulation:** The model provides separate updating equations for the level ($\widehat{l}_t$), trend ($\widehat{b}_t$), and seasonal ($\widehat{s}_t$) components, using smoothing parameters α, β, and γ, respectively. Let L be the length of the seasonal period.

  Additive Method: Used when the seasonal variation is roughly constant throughout the series.

$$\text{Level:} \quad \widehat{l}_t = \alpha(y_t - \widehat{s}_{t-L}) + (1-\alpha)(\widehat{l}_{t-1} + \widehat{b}_{t-1}) \tag{6}$$

$$\text{Trend:} \quad \widehat{b}_t = \beta(\widehat{l}_t - \widehat{l}_{t-1}) + (1-\beta)\widehat{b}_{t-1} \tag{7}$$

$$\text{Seasonality:} \quad \widehat{s}_t = \gamma(y_t - \widehat{l}_t) + (1-\gamma)\widehat{s}_{t-L} \tag{8}$$

  The forecast for h steps ahead is given by:

$$\widehat{\boldsymbol{y}}_{t+h|t} = \widehat{l}_t + h\widehat{b}_t + \widehat{s}_{t-L+h_L^+} \quad \text{where } h_L^+ = \lfloor (h-1) \pmod L \rfloor + 1 \tag{9}$$

  Multiplicative Method: Used when the seasonal variation changes in proportion to the level of the series.

$$\text{Level:} \quad \widehat{l}_t = \alpha\left(\frac{y_t}{\widehat{s}_{t-L}}\right) + (1-\alpha)(\widehat{l}_{t-1} + \widehat{b}_{t-1}) \tag{10}$$

$$\text{Trend:} \quad \widehat{b}_t = \beta(\widehat{l}_t - \widehat{l}_{t-1}) + (1-\beta)\widehat{b}_{t-1} \tag{11}$$

$$\text{Seasonality:} \quad \widehat{s}_t = \gamma\left(\frac{y_t}{\widehat{l}_t}\right) + (1-\gamma)\widehat{s}_{t-L} \tag{12}$$

The forecast for h steps ahead is given by:

$$\widehat{\boldsymbol{y}}_{t+h|t} = (\widehat{l}_t + h\widehat{b}_t)\widehat{s}_{t-L+h_L^+} \quad \text{where } h_L^+ = \lfloor (h-1) \pmod{L} \rfloor + 1 \tag{13}$$

- **Forecasting Type:** A statistical model for point forecasting that can handle various combinations of trend and seasonality.

C.0.15 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) architecture specifically designed to address the vanishing gradient problem, allowing it to learn and remember long-term dependencies in sequential data. Unlike traditional neural networks, LSTMs have internal mechanisms called "gates" that regulate the flow of information. These gates enable the network to selectively remember or forget information over long periods, making it particularly well-suited for time series forecasting. [23]

- **Input:** A sequence of historical time series observations.
- **Output:** A point forecast for one or more future time steps.
- **Mathematical Formulation:** The core of an LSTM unit is its cell state, $\widehat{\boldsymbol{c}}_t$, which acts as a memory. The flow of information into and out of the cell is controlled by three gates: the forget gate ($\boldsymbol{f}_t$), the input gate ($\boldsymbol{i}_t$), and the output gate ($\boldsymbol{o}_t$). At each time step t, these gates update the cell state and produce a hidden state, $\widehat{\boldsymbol{h}}_t$.

$$\text{Forget Gate:} \quad \boldsymbol{f}_t = \boldsymbol{\sigma}(\boldsymbol{w}_f \cdot [\widehat{\boldsymbol{h}}_{t-1}, y_t] + b_f) \tag{14}$$

$$\text{Input Gate:} \quad \boldsymbol{i}_t = \boldsymbol{\sigma}(\boldsymbol{w}_i \cdot [\widehat{\boldsymbol{h}}_{t-1}, y_t] + b_i) \tag{15}$$

$$\text{Candidate State:} \quad \widetilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{w}_c \cdot [\widehat{\boldsymbol{h}}_{t-1}, y_t] + b_c) \tag{16}$$

$$\text{Cell State Update:} \quad \widehat{\boldsymbol{c}}_t = \boldsymbol{f}_t \odot \widehat{\boldsymbol{c}}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}}_t \tag{17}$$

$$\text{Output Gate:} \quad \boldsymbol{o}_t = \boldsymbol{\sigma}(\boldsymbol{w}_o \cdot [\widehat{\boldsymbol{h}}_{t-1}, y_t] + b_o) \tag{18}$$

$$\text{Hidden State Update:} \quad \widehat{\boldsymbol{h}}_t = \boldsymbol{o}_t \odot \tanh(\widehat{\boldsymbol{c}}_t) \tag{19}$$

  where $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weight matrices and bias vectors for each gate, $\boldsymbol{\sigma}$ is the sigmoid function, and $\odot$ denotes element-wise multiplication. The final prediction is typically generated by passing the hidden state $\widehat{\boldsymbol{h}}_t$ through a dense output layer.
- **Forecasting Type:** A neural network model for point forecasting that can capture complex non-linear patterns in time series data.

C.0.16 Prophet

Prophet is a forecasting procedure developed by Meta, based on a decomposable time series model. It is designed to be robust to missing data and shifts in the trend, and it typically handles holidays and seasonal effects well. The model fits an additive model with components for trend, seasonality, and holidays. [26]

- **Input:** A univariate time series with timestamps.
- **Output:** A point forecast, along with uncertainty intervals.
- **Mathematical Formulation:** The Prophet model is specified as a sum of three components:

$$y_t = g(t) + s(t) + h(t) + \epsilon_t \tag{20}$$

  where:
  - $g(t)$ is the trend component, which is modeled as either a piecewise linear or logistic growth function. This allows the model to capture non-periodic changes in the time series.
  - $s(t)$ is the seasonality component, which models periodic changes (e.g., yearly, weekly, daily). It is approximated by a Fourier series:

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n t}{P}\right) + b_n \sin\left(\frac{2\pi n t}{P}\right) \right) \tag{21}$$

    where P is the period of the seasonality (e.g., 365.25 for yearly).

- $h(t)$ is the holiday component, which represents the effects of holidays and special events. It is modeled as a sum of indicator functions for each holiday.
- ϵ_t is the error term, assumed to be normally distributed white noise.

- **Forecasting Type:** A decomposable statistical model for point and probabilistic forecasting, particularly effective for business time series with strong seasonal patterns and holiday effects.

C.0.17 Random Forest

Random Forest is an ensemble machine learning model that operates by constructing a multitude of decision trees at training time. For time series forecasting, it is applied as a regression model to a featurized dataset. By fitting numerous trees on various sub-samples of the data and employing randomness in feature selection, it improves predictive accuracy and controls over-fitting. The final prediction is an average of the outputs from all individual trees, making the model robust and capable of capturing complex, non-linear relationships. [27]

- **Input:** A feature matrix $\boldsymbol{X}$ where rows are observations and columns are engineered features (e.g., lags, calendar variables), and a corresponding target vector $\boldsymbol{y}$.
- **Output:** A point forecast for each input feature vector.
- **Architecture and Formulation:** A Random Forest is an ensemble of B decision trees. Its predictive power comes from two sources of randomness introduced during training:
  - **Bagging (Bootstrap Aggregating):** Each individual tree, f_b, is trained on a bootstrap sample (a random sample drawn with replacement) from the original training dataset.
  - **Feature Randomness:** When splitting a node in a tree, the algorithm considers only a random subset of the total features, which decorrelates the trees in the forest.

  For a new input feature vector $\boldsymbol{x}$, the forecast is the average of the predictions from all B trees in the ensemble:

$$\widehat{\boldsymbol{y}}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} f_b(\boldsymbol{x}) \tag{22}$$

- **Forecasting Type:** An ensemble machine learning model for point forecasting. It is non-parametric and highly effective at modeling non-linear relationships between features and the target variable.

C.0.18 Seasonal Naive

The Seasonal Naive model is a simple yet effective baseline method for forecasting time series with a strong seasonal component. Its core principle is that the forecast for a future period is equal to the last observed value from the same season. For example, the forecast for this Monday would be the value from last Monday. Despite its simplicity, it serves as a crucial benchmark for more complex models. [28]

- **Input:** A univariate time series with a known seasonal period.
- **Output:** A point forecast for future time steps.
- **Mathematical Formulation:** The forecast for h steps ahead from time t, denoted $\widehat{\boldsymbol{y}}_{t+h|t}$, is the last observed value from the corresponding season. Let L be the seasonal period (e.g., $L = 7$ for daily data with weekly seasonality). The forecast is given by:

$$\widehat{\boldsymbol{y}}_{t+h|t} = \boldsymbol{y}_{t+h-L\cdot k} \tag{23}$$

  where $k = \lceil h/L \rceil$ is an integer that ensures the lagged time index refers to the most recent observation from the target season. For a one-season-ahead forecast ($h = L$), this simplifies to $\widehat{\boldsymbol{y}}_{t+L|t} = \boldsymbol{y}_t$.
- **Forecasting Type:** A simple statistical baseline for seasonal point forecasting.

C.0.19 Support Vector Regression (SVR)

Support Vector Regression (SVR) is a supervised learning algorithm that extends the principles of Support Vector Machines (SVMs) to regression problems. Instead of finding a hyperplane that separates classes, SVR aims to find a function that deviates from the target values by a value no greater than a specified margin, ϵ, for as many of the training points as possible. It is particularly effective in high-dimensional spaces and is robust to some outliers due to its use of an ϵ-insensitive loss function, which ignores errors within this margin. [29]

- **Input:** A feature matrix $\boldsymbol{X}$ and a corresponding target vector $\boldsymbol{y}$.
- **Output:** A point forecast for each input feature vector.
- **Mathematical Formulation:** The goal of SVR is to find a function $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ that is as "flat" as possible. This is achieved by minimizing the norm of the weight vector, $||\boldsymbol{w}||^2$. The optimization problem is formulated to tolerate errors up to a margin ϵ while penalizing points that fall outside this margin using slack variables ξ_i and ξ_i^*. The primal optimization problem is:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{24}$$

  subject to the constraints:

$$\boldsymbol{y}_i - (\boldsymbol{w}^T\boldsymbol{x}_i + b) \leq \epsilon + \xi_i \tag{25}$$

$$(\boldsymbol{w}^T\boldsymbol{x}_i + b) - \boldsymbol{y}_i \leq \epsilon + \xi_i^* \tag{26}$$

$$\xi_i, \xi_i^* \geq 0 \tag{27}$$

  where C is a regularization parameter that controls the trade-off between the flatness of the model and the amount up to which deviations larger than ϵ are tolerated. Non-linear relationships are handled by mapping the data to a higher-dimensional space using a kernel function.
- **Forecasting Type:** A machine learning model for point forecasting that is robust to some outliers and effective in high-dimensional feature spaces.

C.0.20 Theta Method

The Theta method is a statistical forecasting technique based on the concept of decomposition. It models a time series by breaking it down into two components, or "theta lines." The first line represents the long-term trend of the data, while the second line is constructed to capture the short-term dynamics by modifying the curvature of the original series. These two lines are forecasted independently and then combined to produce the final forecast. The standard Theta model has been shown to be equivalent to Simple Exponential Smoothing with a drift term. [30]

- **Input:** A univariate time series.
- **Output:** A point forecast for future time steps.
- **Mathematical Formulation:** The method decomposes the original time series, y_t, into two theta lines.
  - **Line 1 (Trend Component):** This line is the simple linear trend fitted to the data, which is found by ordinary least squares regression:

$$\widetilde{\boldsymbol{y}}_t^{(1)} = \widehat{a} + \widehat{b}t \tag{28}$$

    This line is extrapolated linearly to produce its forecast.
  - **Line 2 (Short-term Component):** This line is constructed by modifying the original series with a coefficient θ. A common and effective choice is $\theta = 2$, which doubles the local curvatures of the series. This modified series, $\widetilde{\boldsymbol{y}}_t^{(2)}$, is then forecasted using Simple Exponential Smoothing (SES).

  The final forecast, $\widehat{\boldsymbol{y}}_{t+h}$, is a simple average of the forecasts from the two lines:

$$\widehat{\boldsymbol{y}}_{t+h} = \frac{1}{2}\left(\widehat{\boldsymbol{y}}_{t+h}^{(1)} + \widehat{\boldsymbol{y}}_{t+h}^{(2)}\right) \tag{29}$$

- **Forecasting Type:** A statistical decomposition model for point forecasting, often used as a strong baseline for its simplicity and performance.

C.0.21 XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful and efficient implementation of the gradient boosting framework. It is an ensemble model that builds decision trees sequentially, where each new tree is trained to correct the errors made by the previous ones. For time series forecasting, XGBoost is used as a regression model on a featurized dataset, making it highly effective at capturing complex, non-linear relationships between the engineered features (e.g., lags, calendar variables) and the target. [28]

- **Input:** A feature matrix $\boldsymbol{X}$ and a corresponding target vector $\boldsymbol{y}$.
- **Output:** A point forecast for each input feature vector.
- **Architecture and Formulation:** XGBoost builds an additive model where the final prediction is the sum of the predictions from K decision trees:

$$\widehat{\boldsymbol{y}}_i = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i) \tag{30}$$

  The trees are added one at a time in a greedy fashion. The k-th tree, f_k, is chosen to minimize a regularized objective function:

$$\mathcal{L}^{(k)} = \sum_{i=1}^{n} l(\boldsymbol{y}_i, \widehat{\boldsymbol{y}}_i^{(k-1)} + f_k(\boldsymbol{x}_i)) + \Omega(f_k) \tag{31}$$

  where l is a differentiable loss function, $\widehat{\boldsymbol{y}}_i^{(k-1)}$ is the prediction from the first $k-1$ trees, and Ω is a regularization term that penalizes the complexity of the tree:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{32}$$

  Here, T is the number of leaves in the tree, $\boldsymbol{w}$ is the vector of scores on the leaves, and γ and λ are regularization parameters.
- **Forecasting Type:** An ensemble machine learning model for point forecasting, known for its high performance, speed, and regularization capabilities.

D Benchmark Tasks Included

In this section, we describe the datasets that have been used for each benchmark task. We summarize the dataset used for each benchmark task in Table 5.

D.1 Synthetic Data: Cyclic Seasonality with Additive Trends

D.1.1 Description

This category of synthetic data models a time series that exhibits both a complex seasonal pattern and a persistent, long-term trend. The data is generated using two related methods. Both methods start with a foundational signal that combines multi-frequency sinusoids with a linear trend. The second, more complex method builds upon this foundation by introducing an additional, randomized sinusoidal component to the signal.

In both cases, non-negative noise from an exponential distribution is added to the deterministic signal. These datasets are ideal for testing a model's ability to identify and separate periodicities from an underlying linear trend, with the second method providing a more complex seasonal structure.

D.1.2 Mathematical Formulation

The generation process for both methods is based on a primary signal, $y_{\text{base}}(t)$, which includes seasonal, trend, and offset components:

$$y_{\text{base}}(t) = \underbrace{2\sin(t) + 2\cos\left(\frac{t}{2}\right)}_{\text{Seasonality}} + \underbrace{\frac{1}{4}t}_{\text{Trend}} + \underbrace{4}_{\text{Offset}} \tag{33}$$

Method 1: Fixed Additive Trend In the first method, the true signal, $y_1(t)$, is simply the base function. The final observed value, Y_t, is this signal plus an additive noise term, ϵ_t.

$$Y_t = y_1(t) + \epsilon_t = y_{\text{base}}(t) + \epsilon_t \tag{34}$$

Method 2: Randomized Additive Trend The second method introduces additional complexity. For each generated time series, a random frequency parameter, α, is sampled once from a continuous uniform distribution:

$$\alpha \sim U(a, b) \tag{35}$$

In the provided code, this range is fixed from $a = 0$ to $b = 5$. This parameter is used to create an additional sinusoidal component that is added to the base signal. The true signal, $y_2(t)$, is therefore:

$$y_2(t) = y_{\text{base}}(t) + \sin(\alpha t) \tag{36}$$

The final observed value, Y_t, is this enhanced signal plus the noise term:

$$Y_t = y_2(t) + \epsilon_t \tag{37}$$

Noise Model For both methods, the noise term, ϵ_t, is drawn from an exponential distribution with a scale parameter β:

$$\epsilon_t \sim \text{Exponential}(\beta) \tag{38}$$

D.1.3 Adjustable Parameters

The data generation process is controlled by the following parameters.

- **Number of Points (`num_points`, N):** This integer parameter sets the total number of data points, defining the length of the time series.
- **Start Time (`start_time`, t_0):** This parameter defines the initial time value for the series.
- **Noise Scale (`noise_std`, β):** This parameter represents the scale (and mean) of the exponential noise distribution. A larger value for β increases the average magnitude of the positive noise added to the base signal.

Table 5: Summary of datasets used for benchmark tasks.

Benchmark	Task	l	h	n	m
Trend					
Multivariate (Non-stationary)	Electricity Consumption [31]	512	64	1741	44
Univariate (Non-stationary)	Software Development Job Postings [32]	512	64	1827	1
Decomposition					
Univariate (Additive)	Synthetically Generated Additive (Section D.1)	1024	64	3000	1
Univariate (Multiplicative)	Synthetically Generated Multiplicative (Section D.2)	1024	64	3000	1
Frequency					
Multivariate (Days)	Gold Price in India [33]	1024	64	4024	5
Univariate (Days)	Coinbase Litecoin [34]	512	64	1827	1
Multivariate (Hours)	Madrid Transport Pollution [35]	2048	64	181753	14
Multivariate (Minutes)	Historical Stock Data (2003-2024) [36]	2048	64	122110	6
Multivariate (Minutes)	Historical Stock Data (2003-2024, Longest) [36]	2048	64	122110	6
Multivariate (Months)	Airlines Baggage Complains [37]	32	8	84	4
Univariate (Months)	Inventories to Sales Ratio [38]	64	64	402	1
Univariate (Quarters)	German House Prices [39]	32	32	221	1
Multivariate (Seconds)	Utah Drilling [40]	2048	64	9661	35
Univariate (Weeks)	Federal Funds Effective Rate [41]	1024	64	3713	1
Univariate (Years)	Personal Consumption Expenditures [42]	32	8	96	1
Seasonality					
Multivariate (Periodic)	Madrid Transport (Cyclical) [35]	2048	64	181753	14
Univariate (Periodic)	Synthetic Cyclic	1024	64	3000	1
Univariate (Quasiperiodic)	Synthetic Non-stationary	1024	64	3000	1
Domain					
Univariate (Climate)	Delhi Climate [43]	512	64	1462	1
Multivariate (Economics/Finance)	Gold Price in India [33]	1024	64	4024	5
Multivariate (Economics/Finance)	Gold Price in India (Real) [33]	1024	64	4024	5
Univariate (Economics/Finance)	Coinbase Litecoin [34]	512	64	1827	1
Multivariate (Energy)	Room SplitSmart [44]	2048	64	10603	2
Univariate (Energy)	Room SplitSmart [44]	128	64	561	1
Multivariate (Healthcare)	NYC Covid Cases [45]	512	64	2005	54
Univariate (Healthcare)	Employees Health Care [46]	64	64	427	1
Univariate (Manufacturing)	Inventories to Sales Ratio [38]	64	64	402	1
Multivariate (Nature)	Soil Monitoring [47]	1024	64	4323	127
Multivariate (Nature)	Soil Monitoring (500) [47]	1024	64	4323	127
Univariate (Nature)	Soil Monitoring [47]	128	64	679	1
Multivariate (Sales)	Airlines Baggage Complains [37]	32	8	84	4
Univariate (Sales)	German House Prices [39]	32	32	221	1
Multivariate (Software)	Cyber Attacks on Water Distribution Networks [48]	512	64	1741	44
Univariate (Software)	Software Development Job Postings [32]	512	64	1827	1
Multivariate (Transport)	Airlines Baggage Complains (100) [37]	32	8	84	4
Multivariate (Transport)	Madrid BEN pollution [49]	2048	64	181753	14
Multivariate (Transport)	Madrid BEN pollution (Noisy) [35]	2048	64	181753	14
Univariate (Transport)	Madrid BEN pollution [49]	2048	64	172622	1
Univariate (Web)	Web Traffic [50]	1024	64	2793	1
Data sparsity					
Multivariate (Dense)	Gold Price in India [33]	1024	64	4024	5
Univariate (Dense)	Chicken Pox [51]	128	64	522	1
Univariate (Sparse)	Patient Chart [52]	2048	64	8093	1
Value type					
Univariate (Binary)	Absenteeism at Work [53]	128	64	740	1
Univariate (Categorical)	Online Retail [54]	2048	64	541909	1
Multivariate (Continuous)	Gold Price in India [33]	1024	64	4024	5
Univariate (Continuous)	Forest Fires [55]	128	64	517	1
Multivariate (Count)	Madrid BEN pollution [49]	2048	64	181753	14
Univariate (Count)	Occupancy [56]	2048	64	10129	1

- **Random Frequency (`alpha`, α):** (Method 2 only) This parameter is not set by the user but is sampled internally from a uniform distribution $U(0, 5)$ for each generated series. It introduces variability in the seasonal component across different datasets created by the second method.

D.2 Synthetic Data: Cyclic Seasonality with Multiplicative and Additive Trends

D.2.1 Description

This category of synthetic data models a time series characterized by a complex interaction of seasonal components and trends. A key feature is a multiplicative trend, where the amplitude of one of the seasonal components grows exponentially over time. This is combined with another stable seasonal component and a linear additive trend.

The data is generated using two related methods. The first method uses a fixed, deterministic signal. The second method introduces additional complexity by adding another sinusoidal component with a randomized frequency to the base signal. In both cases, non-negative noise from an exponential distribution is added. These datasets are particularly useful for testing a model's ability to handle heteroscedasticity, where the variance of the series changes over time, in the presence of other seasonalities and trends.

D.2.2 Mathematical Formulation

Both methods are built upon a primary signal, $y_{\text{base}}(t)$, which is a composite of several functions:

$$y_{\text{base}}(t) = \underbrace{e^{t/100}\sin(t)}_{\text{Multiplicative Seasonality}} + \underbrace{3\cos\left(\frac{t}{2}\right)}_{\text{Additive Seasonality}} + \underbrace{\frac{1}{2}t}_{\text{Linear Trend}} \tag{39}$$

Method 1: Fixed Multiplicative Trend In the first method, the true signal, $y_1(t)$, is simply the base function. The final observed value, Y_t, is this signal plus an additive noise term, ϵ_t.

$$Y_t = y_1(t) + \epsilon_t = y_{\text{base}}(t) + \epsilon_t \tag{40}$$

Method 2: Randomized Additive Component The second method adds another layer of seasonality. For each generated time series, a random frequency parameter, α, is sampled once from a continuous uniform distribution:

$$\alpha \sim U(a, b) \tag{41}$$

In the provided code, this range is fixed from $a = 5$ to $b = 10$. The true signal, $y_2(t)$, is the base signal plus this new randomized sinusoidal component:

$$y_2(t) = y_{\text{base}}(t) + \sin(\alpha t) \tag{42}$$

The final observed value, Y_t, is this enhanced signal plus the noise term:

$$Y_t = y_2(t) + \epsilon_t \tag{43}$$

Noise Model For both methods, the noise term, ϵ_t, is drawn from an exponential distribution with a scale parameter β:

$$\epsilon_t \sim \text{Exponential}(\beta) \tag{44}$$

D.2.3 Adjustable Parameters

The data generation process is controlled by the following parameters.

- **Number of Points (`num_points`, N):** This integer parameter sets the total number of data points, defining the length of the time series.
- **Start Time (`start_time`, t_0):** This parameter defines the initial time value for the series.

- **Noise Scale (`noise_std`, β):** This parameter represents the scale (and mean) of the exponential noise distribution. A larger value for β increases the average magnitude of the positive noise added to the base signal.
- **Random Frequency (`alpha`, α):** (Method 2 only) This parameter is not set by the user but is sampled internally from a uniform distribution $U(5, 10)$ for each generated series. It introduces variability in the seasonal component across different datasets created by the second method.

E Benchmark Evaluations

Our benchmark evaluation across multiple deterministic and stochastic forecasting metrics reveals several key insights regarding model performance and efficiency. Notably, LAFN, with only 0.4M (400K) parameters, demonstrates remarkable performance efficiency compared to significantly larger foundation models such as TimesFM (200M parameters) [17], Chronos (20M parameters) [7], and Moirai (approximately 91M for the base variant) [15].

On deterministic metrics, LAFN achieves the best performance on several tasks: it attains the lowest MAE on the Baggage dataset (0.358) and SplitSmart Energy task (0.645) (see Table 6), the lowest RMSE on Baggage (0.483) (Table 9), and the lowest MASE on both Baggage (0.689) and SplitSmart (1.524) (Table 8). This performance persists despite LAFN's compact architecture, achieving competitive or superior results compared to models with up to $2000\times$ more parameters. For instance, while LAFN achieves the best MAE on Baggage, TimesFM (200M parameters) achieves 0.526 on the same task, and Chronos (depending on variant) achieves 0.841. Similarly, on SplitSmart, LAFN's MASE of 1.524 outperforms TimesFM (1.617), demonstrating that parameter efficiency does not necessarily compromise forecasting accuracy.

On stochastic metrics, LAFN also shows strong performance, achieving the best CRPS scores on Madrid Count and Madrid Hours datasets (both 0.798) (Table 12), despite competing against large-scale foundation models. The quantile score (Table 10) and weighted interval score (Table 11) results further validate LAFN's capability to provide well-calibrated probabilistic forecasts with minimal model complexity.

These findings align with recent research on efficient forecasting architectures [57] and suggest that architectural design and training methodology are as important as model scale for time series forecasting [7, 15]. LAFN's success highlights the potential for lightweight yet effective forecasting models suitable for resource-constrained environments, while maintaining competitive performance against much larger foundation models [57].

Table 6: MAE Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
prophet	1.3503	0.1703	0.1703	**1.129e-5**	0.8257	0.8257	0.8257	1.7661	0.7788	0.05313
timesfm	0.5265	0.08253	0.08253	1.631e-4	0.75	0.75	0.75	**0.3798**	0.6845	0.00557
chronos	0.8411	0.09603	0.0961	0.0034	1.0979	1.1446	1.1189	0.5629	0.6915	**0.00555**
moirai	0.6021	0.1127	0.1132	0.00418	4074.9799	—	12.2493	0.4652	0.6824	0.0103
exponential_smoothing	1.56	0.1322	0.1322	0.03214	1.1162	1.1162	1.1162	1.1566	0.7413	0.00588
arima	1.451	0.08869	0.08869	0.0377	0.9674	0.9674	0.9674	1.1532	0.7394	0.00594
croston_classic	1.3627	0.09168	0.09168	0.04007	**0.6607**	**0.6607**	**0.6607**	1.3323	0.7505	0.00595
seasonal_naive	1.2866	0.1089	0.1089	0.04068	0.8156	0.8156	0.8156	1.486	0.7609	0.00604
tiny_time_mixer	0.5614	0.1095	0.1095	0.00738	0.8694	0.8694	0.8694	0.4273	0.6896	0.01328
lstm	0.7253	0.1229	0.08364	0.09176	0.9037	0.9174	0.8942	0.6005	0.6949	0.01797
random_forest	0.7274	0.09343	0.09343	0.08293	0.9408	0.9408	0.9408	0.6318	0.7001	0.0272
moment	1.101	0.1355	0.1341	0.115	0.9741	0.965	0.9727	1.0086	0.712	0.03167
LAFN	**0.3584**	0.1259	0.1259	—	0.8623	0.8623	0.8623	0.5147	**0.645**	0.04306
tabpfn	0.523	0.08237	0.08237	—	1.2051	1.2051	1.2051	0.5751	0.6811	0.04421
svr	0.7274	**0.0807**	**0.0807**	0.1292	0.925	0.925	0.925	0.566	0.6847	0.05803
varmax	1.4483	0.1188	0.1188	—	—	—	—	—	0.763	—
lagllama	0.975	1.0764	0.9712	1.0818	1.0346	0.7924	1.0575	0.4772	0.8107	0.3648
toto	0.6329	—	—	—	—	—	—	—	—	—
theta	1.0456	0.7315	0.7315	3.6515	—	—	—	4.033	0.6483	—
moirai_moe	1.7957	—	—	—	—	—	—	—	—	—

Table 7: MAPE Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
prophet	326.9251	180.2845	180.2845	**5.914e-4**	759.798	759.798	759.798	1977.3261	262.7489	12.6987
timesfm	**141.1057**	98.0174	98.0174	0.03857	711.3051	711.3051	711.3051	220.7421	113.7671	6.929e+8
chronos	312.9438	125.4187	127.6942	0.2912	839.1958	895.2794	828.343	269.6401	123.8572	3.516e+9
moirai	623.2775	81.6825	82.0073	0.5541	1.116e+7	—	1.656e+5	229.2512	109.6988	2.574e+8
tiny_time_mixer	223.935	161.4929	161.4929	1.0542	841.4785	841.4785	841.4785	**162.0339**	121.426	1.050e+9
croston_classic	377.9797	95.3737	95.3737	5.8565	**353.8011**	**353.8011**	**353.8011**	1396.8504	221.4088	**1.5506**
seasonal_naive	234.6152	113.4186	113.4186	5.9475	526.714	526.714	526.714	1676.5477	208.3699	1.5745
exponential_smoothing	517.0076	197.8321	197.8321	4.7063	740.4425	740.4425	740.4425	1430.5323	211.1025	1.792
arima	444.6815	91.3389	91.3389	5.5797	728.6722	728.6722	728.6722	1125.1788	209.3847	1.8021
random_forest	624.1922	92.6297	92.6297	12.185	732.1473	732.1473	732.1473	651.4781	147.4979	11.0236
tabpfn	480.2252	45.1138	45.1138	—	939.0734	939.0734	939.0734	547.5877	113.5734	11.693
lstm	577.3367	87.638	115.776	13.5608	701.1755	667.6615	731.9914	582.6126	131.8695	5.806e+7
moment	216.8776	**32.0207**	**25.6073**	16.8106	807.9779	796.8301	815.8814	889.9217	174.3516	4.780e+4
svr	624.1922	89.9103	89.9103	28.163	732.3681	732.3681	732.3681	397.0512	114.873	125.579
varmax	475.8379	33.7242	33.7242	—	—	—	—	—	239.2184	—
LAFN	205.5096	41.5552	41.5552	—	699.4932	699.4932	699.4932	317.9451	89.0227	1.587e+5
lagllama	262.3012	209.8353	145.1316	74.9169	515.0667	386.8356	627.3051	730.9717	163.071	159.1523
theta	218.2205	381.5479	381.5479	1054.1567	—	—	—	6122.3174	**86.1676**	—
toto	316.7353	—	—	—	—	—	—	—	—	—
moirai_moe	632.1069	—	—	—	—	—	—	—	—	—

Table 8: MASE Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
prophet	2.5965	4.3376	4.3376	**0.02839**	2.7294	2.7294	2.7294	13.2827	1.8398	899.244
timesfm	1.0124	2.1017	2.1017	0.41	2.479	2.479	2.479	**2.8563**	1.6168	94.2659
LAFN	**0.6892**	3.2052	3.2052	—	2.8502	2.8502	2.8502	3.8706	**1.5237**	728.785
tabpfn	1.0057	2.0977	2.0977	—	3.9834	3.9834	3.9834	4.3253	1.609	748.3392
tiny_time_mixer	1.0795	2.7885	2.7885	18.5563	2.8738	2.8738	2.8738	3.2137	1.6291	224.8232
moirai	1.1578	2.8695	2.8826	10.509	1.347e+4	—	40.4898	3.499	1.6119	174.2673
toto	1.2169	—	—	—	—	—	—	—	—	—
lstm	1.3948	3.1305	2.13	230.634	2.987	3.0325	2.9556	4.5162	1.6415	304.1776
random_forest	1.3986	2.3795	2.3795	208.4371	3.1097	3.1097	3.1097	4.7518	1.6538	460.3945
svr	1.3986	**2.0552**	**2.0552**	324.7135	3.0577	3.0577	3.0577	4.2569	1.6174	982.1705
theta	2.0105	18.6298	18.6298	9177.467	—	—	—	30.3313	1.5314	—
chronos	1.6172	2.4457	2.4475	8.5428	3.6289	3.7835	3.6986	4.2335	1.6335	**93.8688**
moment	2.1171	3.4509	3.4155	288.9883	3.2198	3.1898	3.2153	7.5856	1.6818	535.9909
arima	2.79	2.2586	2.2586	94.7489	3.1976	3.1976	3.1976	8.6726	1.7467	100.5149
exponential_smoothing	2.9998	3.367	3.367	80.7743	3.6897	3.6897	3.6897	8.6984	1.751	99.4697
croston_classic	2.6202	2.3348	2.3348	100.7	**2.1839**	**2.1839**	**2.1839**	10.0198	1.7727	100.7
seasonal_naive	2.4739	2.773	2.773	102.25	2.6959	2.6959	2.6959	11.1759	1.7973	102.25
varmax	2.785	3.0245	3.0245	—	—	—	—	—	1.8023	—
lagllama	1.8748	27.4121	24.7327	2718.8557	3.4198	2.6191	3.4956	3.5888	1.9151	6173.8657
moirai_moe	3.4529	—	—	—	—	—	—	—	—	—

Table 9: RMSE Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
prophet	1.4893	0.2411	0.2411	**1.892e-5**	1.291	1.291	1.291	1.933	**2.3175**	0.1627
timesfm	0.669	0.1738	0.1738	1.940e-4	**1.1987**	**1.1987**	**1.1987**	**0.623**	2.3527	**0.00743**
chronos	0.967	0.1911	0.1914	0.00495	1.7918	1.8709	1.7704	1.0103	2.3504	0.05167
moirai	0.9712	0.2037	0.2055	0.00499	1.030e+5	—	154.0111	0.7218	2.3553	0.03645
tiny_time_mixer	0.6816	0.2051	0.2051	0.01059	1.309	1.309	1.309	0.6702	2.3501	0.02331
exponential_smoothing	1.6599	0.2596	0.2596	0.03796	1.5877	1.5877	1.5877	1.368	2.3273	0.01826
arima	1.5465	0.182	0.182	0.04549	1.3808	1.3808	1.3808	1.3067	2.3274	0.01846
croston_classic	1.4597	0.1829	0.1829	0.04758	1.2076	1.2076	1.2076	1.4728	2.3258	0.01894
seasonal_naive	1.6317	0.2179	0.2179	0.0483	1.5016	1.5016	1.5016	1.6287	2.3393	0.01923
lstm	0.8749	0.2134	0.1721	0.1079	1.4227	1.458	1.4236	0.8143	2.3458	0.04982
random_forest	0.8789	0.1831	0.1831	0.09686	1.3707	1.3707	1.3707	0.8463	2.3412	0.06252
svr	0.8789	**0.1507**	**0.1507**	0.131	1.4476	1.4476	1.4476	0.7478	2.3468	0.07317
LAFN	**0.4835**	0.2483	0.2483	—	1.3087	1.3087	1.3087	0.7026	2.3899	0.08154
moment	1.2252	0.2642	0.2622	0.1347	1.4038	1.4048	1.4095	1.1703	2.3353	0.09986
tabpfn	0.6452	0.1607	0.1607	—	1.7946	1.7946	1.7946	0.7316	2.3536	0.1387
varmax	1.5476	0.2451	0.2451	—	—	—	—	—	2.3212	—
lagllama	1.1522	1.1101	1.0398	1.2697	1.9641	1.4825	1.9135	0.6247	2.4138	0.5353
theta	1.1915	0.775	0.775	3.7085	—	—	—	4.4442	2.3882	—
toto	0.9174	—	—	—	—	—	—	—	—	—
moirai_moe	1.8892	—	—	—	—	—	—	—	—	—

Table 10: Quantile Score Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
moirai	1.4656	0.3607	0.3692	**0.0116**	**2.2638**	—	**2.2609**	1.6067	**2.9823**	**0.00498**
chronos	3.1579	**0.3429**	**0.3513**	0.01619	3.4517	3.7626	3.5154	2.1077	3.3491	0.00762
LAFN	1.8731	0.4775	0.4775	—	2.8502	2.8502	2.8502	2.0161	3.1046	0.2372
lagllama	3.1239	3.1333	2.9102	3.5586	3.5871	**2.6475**	3.5033	**1.5105**	3.469	1.1164
toto	**1.3928**	—	—	—	—	—	—	—	—	—
moirai_moe	6.4939	—	—	—	—	—	—	—	—	—

Table 11: Weighted Interval Score Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
moirai	6.4414	1.5728	1.6072	**0.05126**	**9.9283**	—	**9.8992**	6.8191	**11.5368**	**0.02172**
chronos	12.4337	**1.398**	**1.4334**	0.067	14.426	15.7645	14.5568	8.2637	12.2022	0.03079
LAFN	8.3478	2.1039	2.1039	—	12.4137	12.4137	12.4137	8.699	11.7468	1.0216
lagllama	13.528	13.7197	12.7306	15.7445	15.3557	**11.5445**	15.3237	**6.6222**	13.2707	4.8724
toto	**6.128**	—	—	—	—	—	—	—	—	—
moirai_moe	26.8475	—	—	—	—	—	—	—	—	—

Table 12: CRPS Results

Model	Baggage	GoldIndia	IndiaGold	LtStock	Madrid Count	Madrid Hours	Madrid Trans.	Soil	SplitSmart	Utah Mfg.
moirai	0.5905	**0.1316**	**0.1344**	**0.00349**	1.497	—	**0.6881**	0.8374	**2.3426**	**0.00188**
chronos	1.8193	0.1365	0.1395	0.00611	1.1359	1.312	1.2141	1.3624	2.477	0.00448
LAFN	1.3921	0.1651	0.1651	—	**0.7978**	**0.7978**	0.7978	0.775	2.3691	0.08721
lagllama	1.1077	1.0429	0.9192	1.858	1.1399	0.9721	1.4797	**0.6107**	2.5584	0.3673
toto	**0.489**	—	—	—	—	—	—	—	—	—
moirai_moe	4.1489	—	—	—	—	—	—	—	—	—

F Additional Related Works

Classical time-series forecasting began with statistical models that exploit stochastic structure and domain priors, including ARIMA and its Box–Jenkins methodology [58], exponential-smoothing state-space ETS [59], the Theta method [60], and multivariate VAR models [61]. Deep learning methods later advanced accuracy and scale by learning nonlinear temporal dependencies from large corpora: DeepAR [62], N-BEATS [63], DLinear [64], TiDE [65], TFT [66], PatchTST [67], and iTransformer [68]. Probabilistic forecasters further model predictive distributions, e.g., diffusion-based TimeGrad [69], score-based CSDI for imputation and forecasting [70], and conditional-flow GRU-NVP [71].

TSFMs. Inspired by NLP/vision pretraining, TSFMs train on heterogeneous corpora and evaluate in zero/few-shot settings across domains and horizons. Representative models include Moirai [72], Chronos [7], TimesFM [73], Lag-Llama [74], Timer [75], UniTS [76], TTM (Tiny Time Mixers) [77], Moment [22], and multimodal VisionTS [78], TiRex [79]. Collectively, they demonstrate strong zero-shot point and probabilistic accuracy on diverse benchmarks while revealing open challenges at long horizons (error accumulation) and at very high frequencies.

Public datasets and repositories. Public corpora have underpinned progress from statistical to foundation-model eras. The M-competitions (M3 and M4) provided broad univariate benchmarks across domains and frequencies [80, 81], followed by the retail-demand M5 competition [82]. The Monash Time-Series Forecasting Archive curates a large, standardized repository spanning many domains and sampling granularities [83]. Large-scale pretraining/evaluation collections include LOTSA (released with Moirai) [72], the Chronos corpus with in-domain/zero-shot splits [7], and the diverse univariate corpus aggregated in Lag-Llama [74]. Task-focused collections such as the LTSF suite [64] (e.g., ETT datasets) and broader benchmarks like TFB [84] and ProbTS [85] assemble datasets emphasizing horizon length, covariates, and probabilistic outputs. Recent work has also introduced BOOM, a large-scale benchmark of 2,807 real-world multivariate observability time series (350 million points) reflecting challenges such as heavy tails and abrupt regime shifts, and BOOMLET, a smaller uniformly sampled subset (23 million points across 1,627 variates) for scalable experimentation [21].

Evaluation frameworks and benchmarks. Tooling and standardized evaluation have evolved in parallel. Practitioner libraries such as Prophet [86] and sktime [87] offer classical and ML baselines with unified interfaces, while GluonTS [88] and PyTorchTS [89] provide probabilistic deep-learning pipelines. Benchmarking efforts including LTSF [64], BasicTS+ [**?**], TFB [84], and ProbTS [85] compare statistical, deep, and (in some cases) foundation models, but differ in task taxonomies, splits, and leakage controls. Standardized metrics such as MASE [90] and CRPS [91] enable cross-dataset aggregation of point and probabilistic performance, yet consistent pretraining/evaluation protocols and leakage-free large-scale corpora remain key needs for fair TSFM assessment.