Tiny-TSM: Efficiently Training a Lightweight SOTA Time Series Foundation Model

Felix Birkel
Prior Labs *
fpb23@cantab.ac.uk

Abstract

We present Tiny-TSM, a time series foundation model characterized by small scale, economical training, and state-of-the-art performance. It comprises 23M total parameters, trained on a single A100 GPU in less than a week using a new synthetic data generation and data augmentation pipeline (SynthTS). Without any neural architecture search, hyperparameter tuning, or scaling up model size, Tiny-TSM achieves state-of-the-art performance on a wide range of time series benchmark datasets, often outperforming much larger models and even matching the performance of much larger, industrial-scale, likely highly tuned foundation models. Specifically, Tiny-TSM outperforms all other time series foundation models we evaluated on medium- and long-term forecasting tasks under MSE loss, while short-term accuracy is still competitive with state-of-the-art models.

We also introduce a causal input normalization scheme that enables time series models to be trained with dense next-token prediction loss, significantly accelerating convergence speed and reducing training time.

All experiments were conducted on a single A100 GPU, illustrating the practicality of the proposed approach in a resource-constrained setting.

Figure 1: Tiny-TSM's relative MSE performance on the GIFT-Eval-NF benchmark compared to other state-of-the-art time series foundation models.

*Work done as an independent researcher before joining Prior Labs.

Preprint.

1 Introduction

Time series analysis is a key domain for modern data science, with applications in healthcare [Che et al., 2016], economics [Stock and Watson, 2002], climate and environmental sciences [Dimri et al., 2020], energy and power systems [Gasparin et al., 2022, Nabavi et al., 2024], transportation and mobility [Liu et al., 2020a], industrial IoT and monitoring [Liu et al., 2020b, Belay et al., 2023], retail and supply chain management [Makridakis et al., 2021, Sartipi et al., 2025], and astronomy [Kügler et al., 2015, Cabrera-Vives et al., 2024]. A modern deep-learning-based approach for time series prediction has attracted growing interest, with an increasing number of publications on time series foundation models [Ansari et al., 2024, Hoo et al., 2025, Das et al., 2024, Cohen et al., 2024, Auer et al., 2025]. Existing work on time series foundation models follows a variety of training paradigms, including masked modeling, fixed-horizon prediction, and other task-specific objectives.

In this work, we investigate how far one can push the performance of a *small*, compute-efficient time-series foundation model trained primarily on synthetic data. We leverage dense next-token prediction together with multi-target forecasting to maximize training signal, and introduce a causal normalization scheme that enables such dense objectives without test leakage. Our main result is that *Tiny-TSM*, a 23M-parameter encoder-based model trained using our *SynthTS* framework, achieves state-of-the-art performance on a broad suite of benchmark datasets, rivaling much larger models trained on curated sets of real world data.

This technical report focuses on the design choices that make this possible, rather than exhaustive neural architecture search or large-scale hyperparameter tuning: All work was carried out as a one-person side project on a single GPU under tight time and budget constraints.

1.1 Contributions

- We introduce **DART-Norm** (Drift Aware Rolling Timeseries Normalization, section 5), a causal input normalization scheme for time series that enables dense next-token prediction losses without test leakage and exposes normalization drift explicitly to the model.
- We propose **SynthTS** (section 4), a pipeline that generates realistic synthetic time series data and allows for seamless augmentation of real-world time series.
- We showcase the effectiveness of the above proposed methods in **Tiny-TSM**, a small, patched encoder-based foundation model for time series. Tiny-TSM matches or exceeds the performance of substantially more expensive and heavily tuned forecasters, achieving state-of-the-art performance across a broad range of benchmarks.
- We introduce **Stride-Interleaved Forecast Inference** (SIFI, section 7), an inference-time ensemble technique that enables models trained with a fixed maximum context length to exploit longer historical series by forecasting on downsampled views and interleaving their predictions. We also demonstrate the benefit of multivariate feature augmentation, where derived transforms of the target variable are added as additional channels to boost accuracy, especially for originally univariate series.
- We showcase the effectiveness of using a **coarse-grid loss** (section 9) to enable strong medium- and long-term forecasting performance. Indeed, Tiny-TSM outperforms all other foundation models we tested in medium- and long-term forecasting tasks, largely due to coarse-grid loss and SIFI.

2 Background

2.1 Time series forecasting

Let $(x_t)_{t\geq 0}$ denote a univariate time series, and let $x_{a:b}$ denote the subsequence $(x_a, x_{a+1}, \ldots, x_b)$. The time-series forecasting problem is to predict future values given past observations, typically by approximating the conditional distribution

$$p(x_{t+1:T} \mid x_{0:t}). \tag{1}$$

In many practical applications, a *point forecast* suffices, in which case the goal is to estimate the conditional mean

$$\mathbb{E}\big[x_{t+1:T} \mid x_{0:t}\big] \tag{2}$$

Figure 2: Per-prediction-length results on GIFT-EVAL-NF

in case of a mean squared error (MSE) loss, or the corresponding conditional median for a mean absolute error (MAE) loss.

Time series may be univariate ($x_t \in \mathbb{R}\ \forall t$) or multivariate ($x_t \in \mathbb{R}^n\ \forall t$). In multivariate time series forecasting, one often aims to predict a single *target* covariate - the remaining covariates can be useful to inform predictions for the target, but need not be predicted themselves. Let $x_{m,t}$ denote the target variable and $x_{1:n,t}$ the full covariate vector at time t. The corresponding conditional mean in Eq. (2) becomes

$$\mathbb{E}\big[x_{m,t+1:T} \mid x_{1:n,0:t}\big]. \tag{3}$$

More generally, some covariates may be known into the future (e.g., calendar features). Let $I \subset \{1, \ldots, n\}$ index the covariates with known future values. The forecasting problem is then

$$\mathbb{E}\big[x_{m,t+1:T} \mid x_{1:n,0:t}\ ,\ x_{I,t+1:T}\big], \tag{4}$$

This setting includes the previous one as a special case (no future known covariates), hence we focus on this setting for this report.

2.2 Time series foundation models

The foundation model paradigm has received strong attention in recent years, originally popularized by large language models (LLMs) in NLP [Bommasani et al., 2022]. Foundation models are powerful pre-trained models that solve downstream tasks via in-context learning [Brown et al., 2020] and/or light fine-tuning.

The foundation model paradigm has since been extended beyond NLP to computer vision [He et al., 2021, Radford et al., 2021], tabular data [Hollmann et al., 2023, Grinsztajn et al., 2025], and time series modeling [Das et al., 2024, Ansari et al., 2024]. Models such as TabPFN [Hollmann et al., 2023] showcase the potential of foundation models for tabular and time series data, in many cases outperforming specialized models such as gradient boosting machines.

In recent years, many time series foundation models have been proposed, typically relying on either fixed-length forecasting, masked modeling or autoregressive modeling objectives. Other approaches adapt pre-trained LLMs to model time series directly, for example by tokenizing continuous sequences and leveraging existing language backbones for forecasting and representation learning.

Despite rapid progress, many current time-series foundation models are large and expensive to train, and their performance often depends on curated real-world datasets and substantial computational budgets. This motivates the question addressed in this work: *How competitive can a small, synthetic-data-driven foundation model be if equipped with the right normalization, data generation, and inference strategies?*

Figure 3: The Tiny-TSM architecture: We employ a patched encoder-based architecture. First, a linear patch embedding projects patches into the model's hidden dimension. In the encoder stack, we interleave time- and feature-attention blocks at a 2:1 ratio. The internal structure of an attention block is shown on the right. The output of the encoder stack is processed by two forecast heads: A cross-attention head and a linear head, the outputs of which are added together to form the final per-patch predictions.

Figure 4: The Tiny-TSM pipeline. Data is first normalized using DART-Norm, then patched and consumed by the model. The model produces per-patch predictions, all of which are then de-normalized.

3 Related Work

Deep learning methods for time series. Early deep learning methods for time series forecasting focused on sequence models such as recurrent networks and temporal convolutional networks. More recently, purely feedforward architectures such as N-BEATS [Oreshkin et al., 2020] have shown that carefully designed multilayer perceptrons with backward/forward residual links can attain state-of-the-art accuracy on classical benchmarks. Transformer-based models have since become dominant for long-horizon forecasting. PatchTST [Nie et al., 2023] introduced a patched time series Transformer, in which each univariate series is segmented into overlapping patches that serve as tokens, reducing attention complexity. Today, patching is commonplace in transformer-based time series models.

Time series foundation models. The idea of large, pre-trained foundation models applied in a zero-shot manner has recently become popular for time series. Chronos [Ansari et al., 2024] tokenizes continuous values via scaling and quantization and trains T5-style language models autoregressively on time-series "text". TimesFM [Das et al., 2024] adopts a decoder-only Transformer that operates directly on continuous inputs. Moirai [Woo et al., 2024] proposes a masked encoder-based Transformer trained on the LOTSA corpus, with frequency-specific projection layers and any-variate attention. Parallel to these, tabular foundation models such as TabPFN have been successfully adapted to time

Figure 5: Overview of the SynthTS synthetic data generation and augmentation framework. We visualize a simplified example of a single single series sampled from a base generator and how it is modified by univariate expansions, sparse feature mixing, and post-transforms.

series forecasting (TabPFN-TS) by reframing forecasting as a tabular regression problem and using light feature engineering [Hollmann et al., 2023, Hoo et al., 2025].

Synthetic time series data and data augmentation. Synthetic data and data augmentation have emerged as key ingredients for training robust time series and tabular foundation models. TabPFN is trained entirely on synthetic tabular tasks and nevertheless transfers effectively to time series when combined with appropriate features [Hollmann et al., 2023, Grinsztajn et al., 2025]. CauKer [Xie et al., 2025] proposes a synthetic data pipeline based on Gaussian process kernel composition and structural causal models to pretrain classification TSFMs purely on synthetic sequences. Chronos-2 likewise leverages mixtures of real and synthetic datasets in its training.

4 Synthetic Data

Recent work has shown the vast potential of training on carefully designed synthetic data, as demonstrated by the TabPFN family of models [Hollmann et al., 2023, Grinsztajn et al., 2025]. For training Tiny-TSM, we introduce *SynthTS*, a synthetic time series generation and augmentation framework designed to mimic key properties of real-world series while covering a broad range of temporal patterns. Figure 5 illustrates the overall data generation procedure.

At a high level, SynthTS first samples univariate time series from a family of base series generators based on a randomly chosen time-index and then combines these primitive series through a sequence of composition and transformation steps. The same augmentation pipeline can be applied to a mix of sampled synthetic and real-world time series, allowing synthetic and real data to be mixed seamlessly.

SynthTS.

1. Frequency and parameter sampling. Each synthetic series is associated with a time index and a base observation frequency (e.g., hourly). We sample a base frequency and a random start time to construct the time index. We then sample a set of *compatible* frequencies that are integer multiples of the base frequency, assigning higher weights to naturally occurring scales. For example, for an hourly base frequency we assign higher probability to daily (24h) and weekly (168h) multiples, while still allowing other integer multiples. We also allow the time index to be absent, to simulate pure sequence data.

In addition, we sample other batch-level parameters such as sequence length and noise level.

2. Sampling from base generators. We define a collection (~ 50) of *base generators*, each of which samples from a specific class of time series given a set of input parameters. Examples include

a sinusoidal generator (with random phase and amplitude, given a frequency) or a random polynomial generator. We implement a mixture of periodic, polynomial, and noise-based base generators, along with several additional variants. A comprehensive overview with visualizations is provided in the appendix.

3. Generating derived time series. Starting from the base series sampled in step 2, we apply a multi-step augmentation pipeline. Real-world time series can be incorporated simply by appending them to the base pool, provided lengths and frequencies are matched.

We repeat the following steps for a chosen number of augmentation rounds:

3.1 Univariate expansions. For each input time series, we compute a randomly chosen number N ($N \in \{1, \ldots, 5\}$) of univariate transforms. These transforms operate along the time axis and include shifts, kernel smoothing, and autoregressive transforms.

3.2 Sparse linear combinations along the feature-dimension. We sample random weights to form a set of sparse linear combinations of the expansions from step 3.1. Sparsity is crucial: dense mixtures tend to blur patterns and reduce diversity. Optional noise can be added at this stage.

3.3 Post-transforms. Finally, we apply one of many *post-transforms* focusing on pointwise operations and support effects. Examples include

$$x \mapsto \mathrm{ReLU}(\mathrm{MinMaxScaler}(x) - 0.5),$$

to mimic series with a natural floor at zero, or

$$x_1 \mapsto x_1 \cdot \mathrm{MinMaxScaler}(x_2),$$

to model amplitude modulation. Other post-transforms inject missing values, outliers, or random spikes.

The outputs of step 3.3 are appended to our set of input time series, the intermediate outputs from steps 3.1 and 3.2 are discarded.

In practice, we use between 2 and 5 augmentation rounds and subsample a fixed number of time series for each round from the current pool of inputs, in order to avoid an exponential growth in the number of derived series.

5 Input normalization and enabling next-token loss

Time-series and tabular foundation models must handle inputs with widely varying scales. A common solution is to standardize each series (e.g., subtracting its mean and dividing by its standard deviation) before feeding it to the model, and to apply the inverse transform to the model output. In the time-series setting, however, such normalization interacts in important ways with non-stationarity and with dense next-token prediction objectives.

First, many time series datasets exhibit strong non-stationarity, so scaling parameters estimated on the training portion of the series (often the early part) may not generalize well to future points. This is particularly evident in the presence of exponential or otherwise super-linear trends. Rolling standardization can address this, but can also lead to non-identifiability as illustrated in Figure 6.

Second, per-series standard scaling complicates dense next-token training. If normalization statistics are computed using both context and target points, then using those targets in the loss leaks information. Restricting normalization to a fixed prefix (e.g., the first 80% of each series) avoids leakage but sacrifices the dense training signal provided by next-token objectives.

To address these issues, we introduce *DART-Norm*, a causal normalization scheme that enables dense next-token prediction while (i) ensuring strictly causal normalization with no leakage, and (ii) mitigating artifacts of rolling normalization by exposing stepwise changes in the normalization statistics to the model.

Algorithm: DART-Norm

Figure 6: Rolling standard scaling turns a linear trend (shown on the left) into an eventually constant series (shown on the right), destroying identifiability. Therefore, providing the model with normalization drift metrics is essential when using rolling standard scaling.

Figure 7: Comparing a test-at-end setup with a next-token-prediction setup. In the test-at-end setup, the train data can be used to normalize the input sequence. This is not directly possible in the next-token-prediction setup without causing test leakage or the need to do multiple forward passes.

1. Rolling standardization. Given a time series (y_t), we first compute its rolling mean and rolling standard deviation:

$$m_t = \mathrm{mean}(y_{0:t}) \tag{5}$$

$$s_t = \mathrm{std}(y_{0:t}) \tag{6}$$

and define the rolling-standardized series

$$x_t = \frac{y_t - m_t}{s_t}. \tag{7}$$

2. Drift features. Next, we compute two drift metrics that track changes in the normalization statistics over time: a scaled stepwise difference in the mean, and a logarithmic relative stepwise difference in the standard deviation:

$$d_t = \frac{m_t - m_{t-1}}{s_{t-1}} \tag{8}$$

$$r_t = \log\left(\frac{s_t}{s_{t-1}}\right) = \log s_t - \log s_{t-1}. \tag{9}$$

We then concatenate x_t, d_t, and r_t to form the model inputs at each time step.

3. Next-token targets. Finally, we construct next-token prediction targets *anchored* to the most recent in-context normalization. For a given context length T and prediction horizon h, the targets are

$$\frac{y_{T+1:T+h} - m_T}{s_T}. \tag{10}$$

Empirically, DART-Norm enables dense next-token training with strictly causal normalization and we observe a roughly **3**$\times$ speed-up in convergence, as well as stronger final model performance, compared to a masked "test-at-the-end" training setup. We note however that under a sufficiently large computational budget, we would not necessarily expect a model trained with DART-Norm to converge to a better checkpoint than an otherwise identical model trained with a masked test-at-the-end objective, given that there are no benefits to this normalization scheme in fixed-input-length forecasting tasks that are common in benchmarks - the benefits are in enabling dense gradient signals during training.

Simply feeding x_t to the model is not sufficient, since at inference the model must produce forecasts in the original scale of y_t, not x_t (see Figure 6). Unlike raw input time series, the drift features d_t and r_t naturally exhibit comparable scales across different series and require no further normalization beyond optional clipping. Moreover, given (x_t, d_t, r_t), one can reconstruct y_t up to a global affine transform, implying that the model can internally emulate fixed-window normalizations if beneficial.

In online settings where new data points arrive continuously, DART-Norm also makes it easy to incorporate fresh observations into the training context with minimal overhead, allowing for inexpensive updates to the key-value-cache.

6 Model Architecture

We adopt a standard encoder-based patched time series transformer, following the architecture introduced in Toto [Cohen et al., 2024]. We use a patch length of 32 to keep computation fast; due to time constraints, we did not explore alternative patch sizes. On top of the base architecture, we add a few small adaptations:

- We add learnable embeddings for missing values of shape (patch length, hidden size).
- We add a lightweight cross-attention head that attends to known future covariates, in addition to a linear head, producing patch-level forecasts.
- We remove the initial normalization layer (RMSNorm) prior to the first encoder layer, as normalizing all input patches to a constant scale removes valuable patch-level information.
- We prepend a small number of learnable pad tokens to handle short sequences more effectively.
- We use a 2:1 ratio of time-mixing attention layers to space-mixing attention layers, reflecting the stronger temporal coupling in typical forecasting tasks.

A general overview of the model architecture is described below and visualized in figure 3:

1. Patching: For a given patch length (32 in our case), we chunk the inputs into patches of length $patch_len$:

   $$(x_t)_{t=0,\dots,T} \in \mathbb{R} \;\mapsto\; (\bar{x}_{i,j}) \;\; where \;\; \bar{x}_{i,j} = x_{i*32+j} \quad for\; j = 1, ..., 32, \tag{11}$$

   padding or discarding early time steps as needed. Patches for time series models are analogous to tokens in NLP.

   For multivariate series, we patch (and embed) all channels independently.
2. Patch embedding: We embed each patch into the model's hidden dimension using a linear projection. We concatenate the drift features produced by DART-Norm with the input as described earlier, thus our patch embedding projects $3 \times patch_length \mapsto hidden_size$.
3. Encoder stack: We iteratively apply standard post-norm transformer layers, interleaving multi head attention (MHA), residual connections, feedforward networks (FFN) and normalization layers (RMSNorm).

$$RMSNorm \mapsto MHA \mapsto Residual \mapsto RMSNorm \mapsto FFN \mapsto Residual \tag{12}$$

For maximum expressivity, we use SwiGLU activations in the FFNs, where

$$\begin{aligned} SwiGLU(x) &= Swish_{W_3,\beta}(W_1 x + b) \otimes (W_2 x + c) \\ Swish_{W,\beta}(x) &= W(x\sigma(\beta x)) \end{aligned} \tag{13}$$

for learnable matrices $W_{1,2,3}$ and parameters b, c, β. We apply one spatial transformer layer after every two temporal transformer layers. Temporal attention layers are causal, while spatial ones are not.

4. Output layer. To leverage known future covariates (including time-index-derived features) the patch-level encoder hidden representations feed into two lightweight forecast heads whose outputs are summed:

   (i) a linear head mapping each patch representation directly to the forecast horizon, and

   (ii) a small covariate cross-attention head that refines predictions using the known future covariates.

For multivariate time series, we predict all input variables jointly at each time step, which yields lower-variance gradients than univariate forecasting and naturally supports multi target prediction. Known future covariates are not forecast.

We use a point-forecasting setup with a simple Huber loss. One could instead implement a probabilistic forecast head (mixture-of-t-distribution head [Cohen et al., 2024]; quantile forecast head [Auer et al., 2025]) but we did not explore this direction.

7 Inference

Enabled by the model's highly efficient architecture and the resulting very fast run-time, we can use a number of inexpensive test-time enhancements. We exploit both symmetry and the sequential nature of time series to reduce variance and improve accuracy.

Symmetry-based ensembling. We ensemble the model with a mirrored version of the input:

$$\text{output} = \frac{f(x_t) - f(-x_t)}{2}, \tag{14}$$

where f denotes the model. This idea reduces variance of predictions by exploiting the internal non-linearity of the model that contrasts the linear input transform. Additional ensembles can be formed by adding small amounts of noise to the input (e.g., Gaussian noise with standard deviation equal to 1% of the input's standard deviation).

Multivariate feature augmentation. Because Tiny-TSM is trained natively on multivariate inputs, we append transformed versions of the target variable as additional channels at inference time. Examples include signed-square and signed-square-root transforms, as well as kernel-smoothed variants. While the exact choice of transforms has limited impact, we find that adding several highly correlated channels consistently improves performance, likely by effectively increasing the model's representational capacity for the target.

Stride-Interleaved Forecast Inference. Tiny-TSM is trained with a maximum context length of 4096. To exploit longer historical series without changing the model, we introduce a downsample-and-interleave ensemble.

Intuitively, SIFI runs the model on strided, downsampled views of the history (e.g. even timestamps vs odd timestamps) and then interleaves their predictions back onto the original time grid.

Given a series $(x_t)_{t=1}^{T}$ and a stride n, we construct n downsampled series by taking every n-th point with different offsets:

$$x^{(k)} = (x_k, x_{k+n}, x_{k+2n}, \dots), \qquad k = 1, \dots, n.$$

Table 1: Relative MSE and MAE per model and horizon.

Model	Relative MSE				Relative MAE			
	Overall	Short	Medium	Long	Overall	Short	Medium	Long
Tiny-TSM (ours)	**0.541**	0.512	**0.545**	**0.613**	0.731	0.691	0.756	0.802
Chronos-2	0.542	**0.499**	0.566	0.626	**0.697**	**0.656**	**0.728**	**0.767**
Moirai-2	0.588	0.527	0.637	0.688	0.724	0.673	0.770	0.803
TiRex	0.594	0.531	0.637	0.697	0.723	0.674	0.764	0.802
TimesFM-2	0.647	0.580	0.681	0.780	0.780	0.724	0.819	0.899
TabPFN-TS	0.655	0.553	0.721	0.843	0.767	0.703	0.813	0.879
PatchTST	0.729	0.736	0.694	0.748	0.856	0.842	0.856	0.890
Chronos-L	0.847	0.625	1.050	1.188	0.847	0.739	1.033	1.066
Chronos-B	0.900	0.648	1.190	1.200	0.890	0.750	1.064	1.079
NBeats	1.080	0.996	1.112	1.261	0.919	0.984	0.857	0.817

We run the model on each downsampled series to obtain forecasts $\hat{x}^{(k)}$ at the corresponding coarse time indices, and then interleave these forecasts back onto the original time grid:

$$\hat{x}_t = \hat{x}_j^{(k)} \quad \text{for} \quad t = k + jn,$$

This method works well for time series with strong seasonality as well as smooth time series, but less well for time series with strong local dependencies and significant non-smooth local patterns, where skipping time-steps omits substantial information. Empirically, we observe significant reductions in loss on long-context series when applying SIFI.

A schematic is:

$$\{x_t\}_{t=1,2,3,\ldots} \begin{array}{llll} \{x_t\}_{t=2,4,6,\ldots} \mapsto & Model \mapsto & \{\hat{x}_t\}_{t=2,4,6,\ldots} \mapsto & \searrow \\ \{x_t\}_{t=1,3,5,\ldots} \mapsto & Model \mapsto & \{\hat{x}_t\}_{t=1,3,5,\ldots} \mapsto & \nearrow \end{array} \{\hat{x}_t\}_{t=1,2,3,\ldots} \tag{15}$$

8 Evaluation and Benchmarking

We evaluate Tiny-TSM on all non-financial datasets from the GIFT-Eval benchmark [Aksu et al., 2024], a collection of datasets we refer to as GIFT-Eval-NF. We compare against the leading time-series foundation models and staistical models as of the time of writing, using their per-dataset results from the GIFT-Eval git repository: Chronos (Base / Large) [Ansari et al., 2024], Chronos-2 [Ansari et al., 2025], Moirai-2[Liu et al., 2025], TiRex [Auer et al., 2025], TimesFM-2 [Das et al., 2024], TabPFN-TS [Hoo et al., 2025], PatchTST [Nie et al., 2023], N-Beats[Oreshkin et al., 2020]. For each dataset and model, we compute:

1. Mean Relative MSE
2. Mean Relative MAE

where "relative" denotes the ratio between the model's error and that of a seasonal naive baseline. Being a point-forecaster trained using a Huber loss, Tiny-TSM is adapted well to forecasting under squared error but less well for forecasting under other criteria. Results are shown in figures 1 and 2, as well as table 1.

We did not evaluate any models on any datasets or benchmarks other than GIFT-Eval-NF at any point during this project.

9 Training Details

Tiny-TSM is trained with a patch length of 32; a maximum input sequence length of 4096; a batch size of 4; a learning rate of 1×10^{-4} (AdamW); and a maximum forecast horizon of 960 time steps.

To encourage strong performance across both short and long horizons, we subsample a maximum forecast horizon from $\{1, \ldots, 960\}$ for each training batch. On a subset of batches, we further

apply a **coarse-grid loss**, computing the loss only on every n-th predicted time step for $n \in \{1, 2, 4, 8, 16, ..., 128\}$. This balances contributions from slow and fast dynamics and mitigates the dominance of noisy and high-variation series in the loss: A low-frequency sinusoid produces much smaller local errors than a high-frequency sinusoid under standard dense losses, leading to weaker gradients for the former.

All experiments were conducted on a single A100 GPU.

10 Limitations and Extensions

This work is intentionally limited in scope, both budget and time. We do not perform neural architecture search or extensive hyperparameter tuning, and we restrict ourselves to a single model size (23M parameters). Scaling up model size, exploring alternative architecture choices, and performing systematic ablations on hyperparameters would likely yield substantial further gains.

Finally, SIFI is applied only at inference time. Incorporating similar coarse-to-dense interpolation strategies directly into the architecture or training objective could allow the model to exploit long contexts more effectively in a single forward pass.

Our current implementation produces point forecasts trained with a Huber loss. This is well aligned with conditional mean forecasting and yields strong performance under MSE, but is less optimal under other losses. Extending Tiny-TSM with a distributional forecast head (e.g., a mixture-of-t head trained with a negative log-likelihood loss, or a quantile head trained with a pinball-Huber loss) would enable flexible trade-offs between metrics by extracting different statistics (mean, median, quantiles) from the predicted distribution. We hypothesize that such distributional training could also improve point forecasts by providing richer gradient signals.

References

Taha Aksu, Gerald Woo, Juncheng Liu, Xu Liu, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. Gift-eval: A benchmark for general time series forecasting model evaluation, 2024. URL https://arxiv.org/abs/2410.10393.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series, 2024. URL https://arxiv.org/abs/2403.07815.

Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, Mononito Goswami, Shubham Kapoor, Danielle C. Maddix, Pablo Guerron, Tony Hu, Junming Yin, Nick Erickson, Prateek Mutalik Desai, Hao Wang, Huzefa Rangwala, George Karypis, Yuyang Wang, and Michael Bohlke-Schneider. Chronos-2: From univariate to universal forecasting, 2025. URL https://arxiv.org/abs/2510.15821.

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, 2025. URL https://arxiv.org/abs/2505.23719.

Mohammed Ayalew Belay, Sindre Stenen Blakseth, Adil Rasheed, and Pierluigi Salvo Rossi. Unsupervised anomaly detection for iot-based multivariate time series: Existing solutions, performance analysis and future directions. *Sensors*, 23(5):2844, 2023.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022. URL https://arxiv.org/abs/2108.07258.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

G Cabrera-Vives, D Moreno-Cartagena, N Astorga, I Reyes-Jainaga, F Förster, P Huijse, J Arredondo, AM Muñoz Arancibia, A Bayo, M Catelan, et al. Atat: Astronomical transformer for time series and tabular data. *Astronomy & Astrophysics*, 689:A289, 2024.

Zhengping Che, S. Purushotham, Kyunghyun Cho, David A. Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8, 2016. URL https://api.semanticscholar.org/CorpusID:4900015.

Ben Cohen, Emaad Khwaja, Kan Wang, Charles Masson, Elise Ramé, Youssef Doubli, and Othmane Abou-Amal. Toto: Time series optimized transformer for observability, 2024. URL https://arxiv.org/abs/2407.07874.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting, 2024. URL https://arxiv.org/abs/2310.10688.

Tripti Dimri, Shamshad Ahmad, and Mohammad Sharif. Time series analysis of climate variables using seasonal arima approach. *Journal of Earth System Science*, 129(1):149, 2020.

Alberto Gasparin, Slobodan Lukovic, and Cesare Alippi. Deep learning for time series forecasting: The electric load case. *CAAI Transactions on Intelligence Technology*, 7(1):1–25, 2022.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Benjamin Jäger, Dominik Safaric, Simone Alessi, Adrian Hayler, Mihir Manium, Rosen Yu, Felix Jablonski, Shi Bin Hoo, Anurag Garg, Jake Robertson, Magnus Bühler, Vladyslav Moroshan, Lennart Purucker, Clara Cornu, Lilly Charlotte Wehrhahn, Alessandro Bonetto, Bernhard Schölkopf, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. Tabpfn-2.5: Advancing the state of the art in tabular foundation models, 2025. URL https://arxiv.org/abs/2511.08667.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners, 2021. URL https://arxiv.org/abs/2111.06377.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second, 2023. URL https://arxiv.org/abs/2207.01848.

Shi Bin Hoo, Samuel Müller, David Salinas, and Frank Hutter. From tables to time: How tabpfn-v2 outperforms specialized time series forecasting models, 2025. URL https://arxiv.org/abs/2501.02945.

Sven Dennis Kügler, Nikos Gianniotis, and Kai Lars Polsterer. Featureless classification of light curves. *Monthly Notices of the Royal Astronomical Society*, 451(4):3385–3392, 2015.

Boyi Liu, Xiangyan Tang, Jieren Cheng, and Pengchao Shi. Traffic flow combination forecasting method based on improved lstm and arima. *International Journal of Embedded Systems*, 12(1): 22–30, 2020a.

Chenghao Liu, Taha Aksu, Juncheng Liu, Xu Liu, Hanshu Yan, Quang Pham, Doyen Sahoo, Caiming Xiong, Silvio Savarese, and Junnan Li. Moirai 2.0: When less is more for time series forecasting, 2025. URL https://arxiv.org/abs/2511.11698.

Yi Liu, Sahil Garg, Jiangtian Nie, Yang Zhang, Zehui Xiong, Jiawen Kang, and M Shamim Hossain. Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach. *IEEE Internet of Things Journal*, 8(8):6348–6358, 2020b.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m5 competition: Background, organization, and implementation. *International Journal of Forecasting*, 2021. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.07.007. Publisher Copyright: © 2021 The Authors.

Seyed Azad Nabavi, Sahar Mohammadi, Naser Hossein Motlagh, Sasu Tarkoma, and Philipp Geyer. Deep learning modeling in electricity load forecasting: Improved accuracy by combining dwt and lstm. *Energy Reports*, 12:2873–2900, 2024.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023. URL https://arxiv.org/abs/2211.14730.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting, 2020. URL https://arxiv.org/abs/1905.10437.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Timothée Hornek Amir Sartipi, Igor Tchappi, and Gilbert Fridgen. Benchmarking pre-trained time series models for electricity price forecasting. *arXiv preprint arXiv:2506.08113*, 2025.

James Stock and Mark Watson. Macroeconomic forecasting using diffusion indexes. *Journal of Business & Economic Statistics*, 20:147–62, 02 2002. doi: 10.1198/073500102317351921.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers, 2024. URL https://arxiv.org/abs/2402.02592.

Shifeng Xie, Vasilii Feofanov, Marius Alonso, Ambroise Odonnat, Jianfeng Zhang, Themis Palpanas, and Ievgen Redko. Cauker: classification time series foundation models can be pretrained on synthetic data only, 2025. URL https://arxiv.org/abs/2508.02879.

A Appendix / supplemental material

A.1 Base Time series Generators

We group base generators by qualitative pattern and provide visual examples in Figure 8. Each group may contain multiple implementations (e.g., several types of noise).

- Sum-of-sinusoids
- Linear and non-linear trends
- Periodic random walks
- Trends with random resets to zero
- Modular circular re-indexing of existing base generators: given a series $(x_t)_{t=1,\dots,L}$ of length L and an integer k, we construct a new index sequence

$$t'_i = (k \cdot i) \bmod L, \qquad i = 1, 2, \dots, \tag{16}$$

  and define the derived series $(x_{t'_i})_i$.
- Periodic processes with intermittent explosive episodes
- Periodic time series with phase shifts
- Periodic non-negative time series with a floor at zero
- Integer-count time series
- Approximately periodic integer time series
- Mixtures of other periodic time series from periodic Base Generators
- Noise processes (Gaussian noise, random walks, uniform noise, etc.)
- Noisy auto-regressive processes
- Explosive processes: Stochastic processes possessing periodicity and localized episodes of explosive scaling
- Local trend processes

Figure 8: Selected examples for time series generated by different groups of Base Generators. We include 2 examples each, except for noise, which has 4 examples due to its substantial importance.

Figure 9: Selected examples from SynthTS. Unlike the samples from the base generators visualized above, these examples are outputs of the full SynthTS pipeline.