Tiny Time Mixers (TTMs): Fast Pre-trained Models for Enhanced Zero/Few-shot Forecasting of Multivariate Time Series

Vijay Ekambaram
IBM Research
Bangalore, India
vijaye12@in.ibm.com

Arindam Jati
IBM Research
Bangalore, India
arindam.jati@ibm.com

Pankaj Dayama
IBM Research
Bangalore, India
pankajdayama@in.ibm.com

Sumanta Mukherjee
IBM Research
Bangalore, India
sumanm03@in.ibm.com

Nam H. Nguyen
IBM Research
Yorktown Heights, NY, USA
nnguyen@us.ibm.com

Wesley M. Gifford
IBM Research
Yorktown Heights, NY, USA
wmgifford@us.ibm.com

Chandra Reddy
IBM Research
Yorktown Heights, NY, USA
creddy@us.ibm.com

Jayant Kalagnanam
IBM Research
Yorktown Heights, NY, USA
jayant@us.ibm.com

Abstract

Large pre-trained models excel in zero/few-shot learning for language and vision tasks but face challenges in multivariate time series (TS) forecasting due to diverse data characteristics. Consequently, recent research efforts have focused on developing pre-trained TS forecasting models. These models, whether built from scratch or adapted from large language models (LLMs), excel in zero/few-shot forecasting tasks. However, they are limited by slow performance, high computational demands, and neglect of cross-channel and exogenous correlations. To address this, we introduce Tiny Time Mixers (TTM), a compact model (starting from 1M parameters) with effective transfer learning capabilities, trained exclusively on public TS datasets. TTM, based on the light-weight TSMixer architecture, incorporates innovations like adaptive patching, diverse resolution sampling, and resolution prefix tuning to handle pre-training on varied dataset resolutions with minimal model capacity. Additionally, it employs multi-level modeling to capture channel correlations and infuse exogenous signals during fine-tuning. TTM outperforms existing popular benchmarks in zero/few-shot forecasting by (4-40%), while reducing computational requirements significantly. Moreover, TTMs are lightweight and can be executed even on CPU-only machines, enhancing usability and fostering wider adoption in resource-constrained environments. The model weights for reproducibility and research use are available here, while enterprise-use weights under the Apache license can be accessed as follows: the initial TTM_Qvariant here, and the latest variants (TTM_B, TTM_E, TTM_A) weights are available here. The source code for the TTM model along with the usage scripts are available here.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

1 Introduction

Multivariate time-series (TS) forecasting entails predicting future values for multiple interrelated time series based on their historical values. The channels[1] being forecast are called target variables, while those influencing the forecasts are termed exogenous variables. This field has seen significant advancements through the application of statistical and machine learning (ML) methods across various domains such as weather, traffic, retail, and energy.

Related Work: Recent advances in multivariate forecasting have been marked by the advent of Transformer-based [31] approaches, exemplified by models like PatchTST [22], Autoformer [38], and FEDFormer [45]. These models have demonstrated notable improvements over traditional statistical and ML methods. Furthermore, architectures based on MLP-Mixer [30], such as TSMixer [6] and TimeMixer [33], have emerged as efficient Transformer alternatives, boasting 2-3X reduced compute requirements with no accuracy compromise compared to their Transformer counterparts.

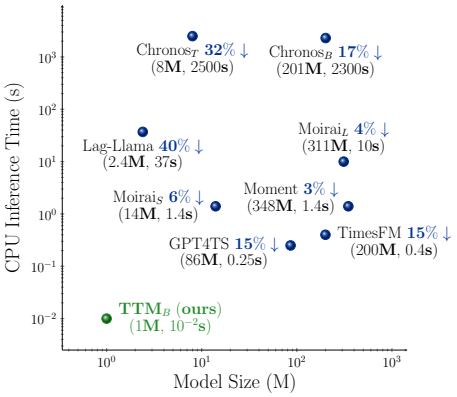

Figure 1: **Size, time, and accuracy overview** of TTM$_B$ vs. open-sourced pre-trained TS benchmarks[2]. We plot each model based on its model size and per batch CPU inference time. The X% mentioned for each baseline indicates that the baseline's forecast is X% less accurate compared to TTM's forecast. Full details in Tables [1–5].

Recently, there has been substantial interest in the research community to build general pre-trained or foundation models (FMs) for TS forecasting that can successfully transfer the learning to unseen target TS dataset, similar to the successes in NLP and vision tasks. However, pre-training in the TS domain is particularly challenging due to the limited public availability of the datasets and the diverse nature across applications. Early in 2024, this interest culminated in the release of several "large" and "massive" TS pre-trained models for forecasting, generating considerable excitement in the research community. Among these releases were Moment[10][3], TimesFM [3][3], Chronos[2][3], Moirai[35][3], and Lag-llama[26][3] that successfully established strong benchmarks in zero-shot forecasting. In addition, there has been a trend towards leveraging pre-trained large language models (LLMs) for TS forecasting, treating the forecasting task as a form of cross-domain transfer learning. These universal cross-transfer approaches, exemplified by recent works such as LLMTime [11], Time-LLM [15], and GPT4TS [46], exhibit promising outcomes in zero/few-shot forecasting scenarios. However, most of these "large" TS pre-trained models demand extremely high computational resources, given their scale ranges from several hundred million to billions of parameters. Given the recent surge in popularity of "small" language models[1][29][39] that address practical resource and cost constraints in real-world industrial settings, this work considers the following question: *Can "tiny" pre-trained models succeed in the TS domain too? If so, can they outperform the zero/few-shot forecasting results of "large" TS pre-trained models demanding significant computational resources and runtime?* Surprisingly, as we demonstrate in this work, the answer is *yes*.

Toward this, we propose **Multi-level Tiny Time Mixers (TTM)**, a significantly smaller pre-trained model (starting from 1 million (M) parameters) for effective zero/few-shot multivariate forecasting. In particular, TTM supports channel correlations and exogenous signals, which are critical and practical business requirements in the context of multivariate forecasting, features lacking in many existing TS pretrained models. TTM is based on the light-weight TSMixer architecture that uses MLPMixer blocks interleaved with simple gated attention as alternatives to the quadratic time-consuming self-attention blocks in Transformers, which makes TTM pre-training and fine-tuning extremely fast. TTM is pre-trained using multiple public datasets (∼1 billion (B) samples) from the Monash and LibCity data repositories. Note that the datasets exhibit considerable diversity in characteristics, such as different domains, temporal resolutions[4] (ranging from seconds to days),

[1] "Channel" refers to an individual dimension in multivariate data (i.e., multivariate or multichannel).

[3] Work done concurrently with this research.

[3] Time-LLM and LLMTime are excluded here as we couldn't run them on CPUs, but their accuracy is compared later in experiments.

[4] Resolution refers to the sampling rate of the input time series (e.g., hourly, 10 minutes, 15 minutes, *etc.*)

lengths, and numbers of channels. Pre-training on such heterogeneous datasets using extremely small models requires specialized architectural advancements. Hence, TTM proposes the following enhancements to the TSMixer architecture for resource-constrained pre-training/fine-tuning: (i) **adaptive patching (AP)** across layers, considering the varied suitability of patch lengths for different datasets, (ii) **diverse resolution sampling (DRS)** to augment the data for increasing coverage across different resolutions, (iii) **resolution prefix tuning (RPT)** to explicitly embed resolution information in the first patch, facilitating resolution-conditioned modeling while training on diverse datasets. Moreover, our approach leverages **multi-level modeling**, where TTMs are first pre-trained in a channel-independent way, and then fine-tuned with channel mixing to incorporate correlations across targets and exogenous channels in the target domain.

Outline of TTM's key capabilities: (1) Amidst the prevalence of "large" pre-trained models demanding significant compute and training time, our work is the first to demonstrate the power of transfer learning using "tiny" TS pre-trained models for zero/few-shot forecasting. (2) Pre-training tiny models on heterogeneous multi-resolution datasets with extremely limited model capacity is challenging. Towards this, we propose various **architectural and training enhancements**, such as **AP**, **DRS**, and **RPT** for robust and resource-constrained pre-training/fine-tuning workflows (as defined above). (3) TTM employs a **multi-level modeling strategy** to explicitly model channel correlations, and incorporate exogenous signals – a crucial capability lacking in most of the existing pre-trained models. (4) Through extensive evaluation of zero/few-shot forecasting on 11 datasets, we establish that TTM models, with sizes as small as 1M parameters, consistently outperform the forecasts of "large" TS pretrained models while offering significant computational benefits. Figure 1 highlights that TTM outperforms popular benchmarks in all three primary dimensions: size, runtime, and accuracy. (5) Given their compact size, zero-shot inference and fine-tuning of TTM models can be easily executed with just one GPU or in CPU-only environments. This greatly enhances the practical adoption and extended reach of our pre-trained models with ease of use.

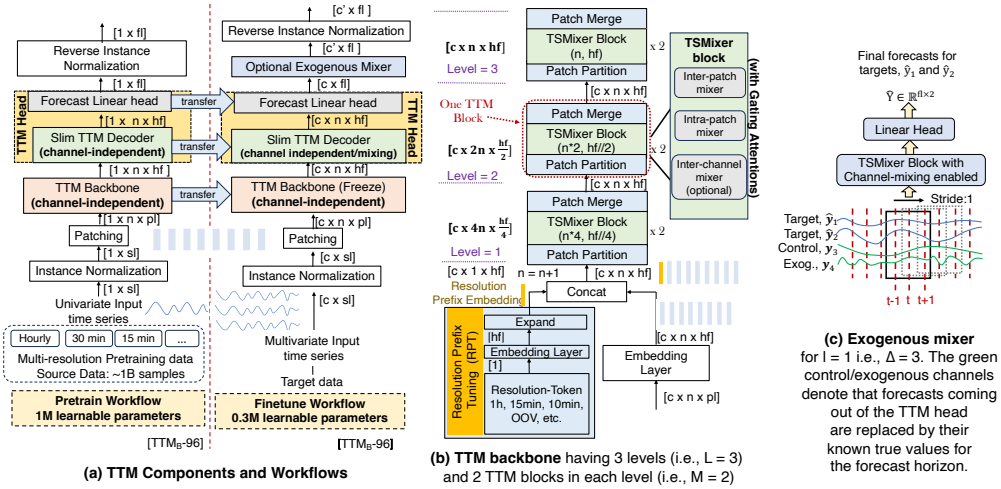

Figure 2: TTM overview (a) Refer to Sections 2 and 3, (b) Refer to Section 3.1, (c) Refer to Section 3.2

2 TTM Components

Let $\boldsymbol{X} \in \mathbb{R}^{c \times sl}$ be a multivariate time series of length sl and number of channels c. The forecasting task can be formally defined as predicting the future values $\boldsymbol{Y} \in \mathbb{R}^{c' \times fl}$ given the history/context $\boldsymbol{X}$. Here, fl denotes the forecast length/horizon, and c' denotes number of forecast channels, where $c' \leq c$. The predictions from the model are denoted by $\hat{\boldsymbol{Y}} \in \mathbb{R}^{c' \times fl}$. In a general multivariate forecasting task, each channel falls into one of the following categories: (a) **Target variables (mandatory):** channels for which forecasts are required, (b) **Exogenous variables (optional):** channels influencing the targets, with known or estimated values throughout the forecast horizon.

2.1 Multi-level Modeling: TTM follows a multi-level architecture consisting of four key components (see Figure 2(a)): (1) The **TTM backbone** is assembled using building blocks derived from the efficient TSMixer architecture [6]. TSMixer is based on MLP blocks interleaved with gated

attention, that enable the mixing of features within patches, across patches and channels, surpassing existing Transformer-based TS approaches with minimal computational requirements. Since TSMixer was not designed to handle multi-resolution data with limited capacity, we introduce various novel enhancements to it as explained later. (2) **TTM decoder** follows the same backbone architecture but is considerably smaller in size, approximately 10-20% of the size of the backbone, (3) **Forecast head** consists of a linear head designed to produce the forecast output, and (4) Optional **Exogenous mixer** serves to fuse exogenous data into the forecasting process. The TTM decoder and forecast head together constitute the **TTM head**, whose weights get updated during the fine-tuning process. This multi-level model refactoring is required to dynamically change the working behavior of various components based on the workflow type, as explained in Section 3. In addition to the above primary components, there is also a preprocessing component as explained next.

2.2 Preprocessing: As shown in Figure 2(a) with colorless blocks, the historical time series $\boldsymbol{X}$ is first **normalized** per instance to have zero mean and unit standard deviation for each channel, to tackle any possible distribution shifts [22, 6]. This process is reversed at the end before computing the loss. The normalized data $\overline{\boldsymbol{X}}$ is subsequently **patched**, $\boldsymbol{X}_p \in \mathbb{R}^{c \times n \times pl}$, into n non-overlapping windows, each of length pl, and then passed to the TTM backbone. Patching, as introduced in [22], has proven to be highly valuable for forecasting. Its effectiveness lies in preserving local semantic information, accommodating longer history, and reducing computation.

3 TTM Methodology

3.1 Pre-training Workflow: TTM operates in two stages: pre-training and fine-tuning (Figure 2(a)). In the pre-training stage, we train the model on a large collection of diverse public datasets. Since the primary focus of TTM is forecasting, pre-training is modeled with a direct forecasting objective. TTM is first pre-trained in a univariate fashion with independent channels on all the existing datasets. Due to varied channel counts in pre-training datasets, modeling multivariate correlations is challenging here; it is addressed later during the fine-tuning stage. Multivariate pre-training datasets are initially transformed into independent univariate TS $(\boldsymbol{X}_1, \cdots, \boldsymbol{X}_N) \in \mathbb{R}^{c(=1) \times sl}$. These are pre-processed (Section 2.2), and subsequently fed into the TTM backbone for multi-resolution pre-training. The output of the backbone $\boldsymbol{X}_h^L \in \mathbb{R}^{(c=1) \times n \times hf}$ is passed through the decoder and forecast head to produce the forecast $\hat{\boldsymbol{Y}} \in \mathbb{R}^{(c=1) \times fl}$ which is then reverse-normalized to return to the original scale. We pre-train the TTM with mean squared error (MSE) loss calculated over the forecast horizon: $\mathcal{L} = ||\boldsymbol{Y} - \hat{\boldsymbol{Y}}||_2^2$. Thus for a given input context length sl and forecast length fl, we get a pre-trained model capturing the common temporal forecasting dynamics and seasonal patterns as observed in the overall pre-training data.

3.1.1 Multi-Resolution Pre-training via TTM Backbone: In TTM, our goal is to create models that are extremely small yet capable of generalizing well across a wide range of diverse datasets with varying resolutions. This is a significant challenge because the models can easily under-fit due to their small size. To tackle these challenges of resource-constrained pre-training, we introduce the following enhancements to the TSMixer backbone.

Adaptive patching (AP): The TTM backbone is crafted with an adaptive patching architecture where different layers of the backbone operate at varying patch lengths and numbers of patches. Since each dataset in the pre-training corpora may perform optimally at a specific patch length, this approach greatly aids in generalization when the pretraining datasets with different resolutions are introduced. Moreover, it helps in scenarios when the availability of the pre-training data is limited as adaptive patching quickly generalizes the model across different granularities. As shown in Figure 2(b), the patched data $\boldsymbol{X}_p \in \mathbb{R}^{c \times n \times pl}$ is passed through a embedding layer to project it to the patch hidden dimension, $\boldsymbol{X}_h \in \mathbb{R}^{c \times n \times hf}$. If the resolution prefix tuning module is activated (as explained later), the resolution prefix is concatenated with $\boldsymbol{X}_h$. For notational simplicity, we denote the concatenated tensor with $\boldsymbol{X}_h$ as well. The TTM backbone consists of L levels, each comprising M TTM blocks with identical patch configurations. The first block in the first level receives $\boldsymbol{X}_h$. The first TTM block in the i-th level, $i = 2, \ldots, L$, receives the processed data $\boldsymbol{X}_h^{(i-1)} \in \mathbb{R}^{c \times n \times hf}$ from the previous block. Each TTM block is further comprised of a patch partition block, a vanilla TSMixer block, and a patch merging block. The patch partition block at level i increases the number of

patches by a factor of K_i and reduces the patch dimension size by the same factor by reshaping $\boldsymbol{X}_h^{(i-1)} \in \mathbb{R}^{c \times n \times hf}$ to $\boldsymbol{X}_h^i \in \mathbb{R}^{c \times (n \cdot K_i) \times (hf/K_i)}$, where $K_i = 2^{(L-i)}$. Figure 2(b) shows the TTM backbone for $L = 3$ and $M = 2$. Note that, we set $hf = m \cdot 2^{L-1}$ for some integer m. Then, TSMixer is applied to the adapted data $\boldsymbol{X}_h^i$. Finally, the output from TSMixer is again reshaped to its original shape (i.e., $\mathbb{R}^{c \times n \times hf}$) in the patch merging block. In subsequent layers, for each increment in level i, the number of patches is halved and the patch dimension doubled. This enables better generalization for small models as we pre-train across multiple datasets. The idea of adaptive patching is popular and very successful in the vision domain (e.g., Swin Transformers [20]) and we successfully apply it to the TS domain to resolve multi-resolution issues in modelling diverse TS datasets. Note that adaptive patching is enabled only in the backbone and not in the decoder, which is designed to be very lightweight.

Augmentation via diverse resolution sampling (DRS): A significant challenge in TS pre-training datasets is the scarcity of public datasets with diverse resolutions. Generally, high-resolution datasets will account for a larger fraction of the samples given their finer sampling resolution. Without adjustment to the training strategy, this can lead to a model that is biased toward the finer resolution data. To overcome this, different strategies are applied to high-resolution datasets to balance the volume of samples at lower resolutions and lead to more uniform coverage. Strategies used include: 1) averaging k samples in sequential, non-overlapping windows to produce a lower resolution dataset; and 2) conventional decimation where only every kth sample is retained. In both cases, the integer k is chosen to achieve the desired resolution from the resolution of the base dataset. For example, from a 4-second resolution dataset, we derive multiple datasets at minutely ($k = 15$) and hourly resolutions ($k = 900$). Note that the original high-resolution dataset remains within the pool of pre-training datasets. This methodology increases the number of datasets for each resolution which greatly improves the model performance.

Resolution prefix tuning (RPT): This technique explicitly learns and incorporates a new patch embedding as a learnable prefix into the input data based on the input resolution (see Figure 2(b) and Table 8). Similar to the concept of prefix tuning [16], this approach provides an explicit signal to the model about the resolution for resolution-conditioned modeling. First, we map every resolution to a unique integer, which is then passed through an embedding layer to project it to the hidden dimension, hf. Subsequently, we expand the embedding across all channels to have a representation of shape $c \times 1 \times hf$. This resolution-based learnable embedding is particularly beneficial in quickly modeling huge volumes of diverse resolution datasets with limited modelling capacity, as the model can easily decouple the data from different resolutions for resolution-conditioned modeling. In addition, RPT also helps in scenarios when the context length (sl) is short. In these scenarios, automatically detecting the resolution becomes a challenge for the model. Hence, by explicitly fusing the resolution information as a prefix, we can enhance the model's ability to learn effectively across resolutions without increasing its size.

3.2 Fine-tuning Workflow: In the fine-tuning workflow, we work with data from the *target* domain that has no overlap with the pre-training datasets. We have three options here: (a) In **zero-shot** forecasting, we directly use the pre-trained model to evaluate on the *test* part of the target data; (b) In **few-shot** forecasting, we utilize only a tiny portion (5-10%) of the *train* part of the target data to quickly update the pre-trained weights of the *TTM head*, and subsequently evaluate it on the *test* part; (c) In **full-shot** forecasting, we fine-tune the pre-trained weights of the *TTM head* on the entire *train* part of the target data, and then evaluate on the *test* part.

The backbone is frozen during fine-tuning, and still operates in a channel-independent univariate fashion. However, the slim decoder in the TTM Head can be fine-tuned utilizing channel mixing or channel independence for multivariate or univariate target data, respectively. If pure multivariate modeling is needed, then the channel-mixer block in all the TSMixer components (see Figure 2(b)) in the decoder is enabled to explicitly capture the cross-channel correlations. The forecast head and reverse normalization perform similar operations as in the pre-training stage. The fine-tuning also optimizes the forecasting objective with MSE loss. This thoughtful multi-level design choice ensures that our backbone excels in channel-independent pre-training, enabling effective temporal correlation modeling across diverse datasets. Simultaneously, the decoder handles target-data-specific tasks like channel-correlation modeling and fine-tuning. In addition, if the target data has exogenous variables, then an exogenous mixer block is applied to the actual forecasts as explained next.

Exogenous Mixer Block: As described in Section 2, the future values of the exogenous channels are known in advance. Let the forecast from the forecast head be $\hat{\boldsymbol{Y}} \in \mathbb{R}^{c \times fl}$. Let the channels $\boldsymbol{x}_0, \cdots, \boldsymbol{x}_{c'}$ denote the target variables and $\boldsymbol{x}_{c'+1}, \cdots, \boldsymbol{x}_c$ denote all exogenous variables with their future values known. First, we replace the forecast values for the exogenous channels with the *true* future values ($\boldsymbol{Y}$) and transpose it: $\hat{\boldsymbol{Y}}_e = [\hat{\boldsymbol{y}}_0, \cdots, \hat{\boldsymbol{y}}_{c'}, \boldsymbol{y}_{c'+1}, \cdots, \boldsymbol{y}_c] \in \mathbb{R}^{fl \times c}$. Next, to learn inter-channel *lagged* correlations, we patch $\hat{\boldsymbol{Y}}_e$ into a series of overlapped windows (i.e., patching with stride$= 1$) to create a new tensor: $\hat{\boldsymbol{Y}}_{e,p} \in \mathbb{R}^{fl \times \Delta \times c}$, where $\Delta = 2 \cdot l + 1$ with l being the context length to incorporate on either side of a time point[5]. Subsequently, we pass $\hat{\boldsymbol{Y}}_{e,p}$ through a vanilla TSMixer block with channel mixing enabled. Thus, the lagged dependency of the forecasts for the target channels on the exogenous channels is seamlessly learned. Finally, we attach a linear head to produce the forecasts for the target channels which is then reshaped as $\hat{\boldsymbol{Y}} \in \mathbb{R}^{c' \times fl}$. Thus, TTM easily handles exogenous infusion which is a practical requirement in any industrial forecasting problem. Figure 2(c) depicts this procedure.

4 Experiments and Results

4.1 Datasets & Metrics : Pre-training employs a subset of ∼1B samples from Monash [9] and Libcity [32] data collection. We specifically exclude a few datasets (like yearly, monthly) as they do not possess sufficient length for the long-term forecasting task. Moreover, we remove all the datasets that we utilize for evaluation (i.e., Weather, Electricity, and Traffic). For zero/few-shot evaluation we consider seven public datasets (**D1**): ETTH1, ETTH2, ETTM1, ETTM2, Weather, Electricity, and Traffic as popularly used in most prior state-of-the-art (SOTA) works [44, 22]. Since these datasets do *not* contain any exogenous variables nor exhibit cross-channel correlation benefits, we incorporate four other datasets (**D2**) for separately validating the efficacy of the decoder channel mixing and exogenous mixer module: bike sharing (BS) [7], carbon capture plant (CC) [13], and 2 more datasets, Application (APP) and Service (SER), from Business and IT observability domain [27, 24]. For full details, refer to the Appendix C. We use mean squared error (MSE) as the standard error metric. In addition, we use the following relative improvement metrics: (i) forecast improvement percentage (*f-imp(%)*) which refers to the MSE (%) improvement of TTM over the considered baseline, averaged across all datasets, and (ii) size improvement metric (*s-imp(X)*) is calculated as the ratio of the baseline model size to the TTM model size (i.e., total parameters).

4.2 SOTA Benchmarks: We benchmark[6] TTM with 24 of the latest open-sourced SOTA forecasting models categorized as follows: (a) **TS pre-trained models:** Lag-Llama [26], TimesFM [3], Moirai [35], Chronos [2] and Moment [10]. (b) **LLM-based TS pre-trained models:** GPT4TS [46], LLMTime [11], Time-LLM [15], UniTime [18] (c) **Self-supervised pre-trained models**: SimMTM [5],Ti-MAE [17], TST [42], LaST [34], TF-C [43], CoST [36] and Ts2Vec [40] (d) **Other architectures:** PatchTST [22], TSMixer [6], TimeMixer [33], iTransformer [19], DLinear [41] and TimesNet [37], FEDFormer [45] and Autoformer [38].

4.3 TTM Model Details: We pre-train three primary variants of TTM as follows: (i) **TTM-Base ($\textbf{TTM}_{\boldsymbol{B}}$)**: 1M parameter model trained with context length, $sl = 512$ and patch length, $pl = 64$, (ii) **TTM-Enhanced ($\textbf{TTM}_{\boldsymbol{E}}$)**: 4M parameter model trained with $sl = 1024$ and $pl = 128$, (iii) **TTM-Advanced ($\textbf{TTM}_{\boldsymbol{A}}$)**: 5M parameter model trained with $sl = 1536$ and $pl = 128$. These TTMs are pre-trained using the 1B pre-training dataset, which takes only 24-30 hours with 6 A100 GPUs, a notably faster time compared to existing counterparts which often take days to weeks. Additionally, for secondary studies, we utilize **Quick TTM ($\textbf{TTM}_{\boldsymbol{Q}}$)**, a variant trained on a smaller subset of the Monash dataset (∼250 million samples), requiring only 4-6 hours for pre-training.

Although, a TTM model needs to be pre-trained for a specific forecast length (FL), we provide two forecast length adaption (FLA) techniques (explained in Section 4.7) that enable a pre-trained TTM to work across different FLs. Users can either build a direct pre-trained model (from one of the above variants) targeting a specific FL, or use the FLA techniques to adapt an existing TTM model to their

[5]This needs padding $\hat{\boldsymbol{Y}}_e$ with zeros of length l on both sides.

[6]For all tables, we highlight the best and second best models with **bold** and underline, respectively. We denote TTM's improvement and degradation *w.r.t.* a baseline with ↑ and ↓ respectively.

application setting. Primary results are reported using the direct approach, and a detailed ablation study is provided to compare the effectiveness of various FLA techniques. In the direct approach, model parameter size varies across FLs and we report the average parameter size in the result tables. TTM fine-tuning and inferencing are highly efficient and fast, requiring only 1 GPU or even CPU execution. All model hyperparameters are chosen based on validation performance, and final test results are reported. Refer to Appendix D for detailed model specifications and hyper-parameters.

Data	TTM_B	TTM_E	TTM_A	Moirai_S	Moirai_B	Moirai_L	TimesFM
ETTH1	**0.394**	0.404	0.4	0.4	0.434	0.51	0.479
ETTH2	0.345	0.335	**0.333**	0.341	0.346	0.354	0.403
ETTM1	0.386	0.38	**0.362**	0.448	0.382	0.39	0.429
ETTM2	0.281	0.271	**0.252**	0.3	0.272	0.276	0.334
Weather	0.237	0.238	**0.231**	0.242	0.238	0.26	-
Electricity	0.205	0.194	0.192	0.233	**0.188**	**0.188**	-
Size	**1M**	**4M**	**5M**	**14M**	**91M**	**311M**	**200M**
TTM_B *f-imp(%) s-imp(X)*				**6% ↑ 14X ↑**	**1% ↓ 91X ↑**	**4% ↑ 311X ↑**	**15% ↑ 200X ↑**
TTM_E *f-imp(%) s-imp(X)*				**7% ↑ 4X ↑**	**1% ↑ 23X ↑**	**6% ↑ 78X ↑**	**16% ↑ 50X ↑**
TTM_A *f-imp(%) s-imp(X)*				**10% ↑ 3X ↑**	**4% ↑ 18X ↑**	**9% ↑ 62X ↑**	**19% ↑ 40X ↑**

Table 1: **Zero-shot** forecast-improvement *(f-imp)* and model size-improvement *(s-imp)* of TTM over Moirai (ICML'24) and TimesFM (ICML'24). MSE averaged across $FL \in \{96, 192, 336, 720\}$. Electricity and Weather results for TimesFM are not reported as its used by TimesFM for pretraining. Similarly, Traffic was used in pre-training for both Moirai and TimesFM. Full table in Appendix F.2

Data	TTM_B	TTM_E	TTM_A	Chronos_T	Chronos_S	Chronos_B	Chronos_L	Lag-llama
ETTH1	**0.204**	0.227	0.214	0.311	0.302	0.252	0.266	0.334
ETTH2	**0.131**	0.151	0.162	0.177	0.16	0.164	0.155	0.168
ETTM1	0.206	0.239	**0.19**	0.839	0.486	0.49	0.538	0.842
ETTM2	0.124	0.128	**0.117**	0.206	0.174	0.19	0.187	0.308
Weather	0.039	0.032	0.043	0.043	0.046	**0.03**	0.033	0.126
Electricity	**0.335**	0.351	0.349	0.423	0.377	0.344	0.339	0.393
Traffic	0.246	**0.24**	0.244	0.291	0.3	0.28	0.269	0.243
Size	**1M**	**4M**	**5M**	**8M**	**46M**	**201M**	**709M**	**3M**
TTM_B *f-imp(%) s-imp(X)*				**32% ↑ 8X ↑**	**26% ↑ 46X ↑**	**17% ↑ 201X ↑**	**18% ↑ 709X ↑**	**40% ↑ 3X ↑**
TTM_E *f-imp(%) s-imp(X)*				**30% ↑ 2X ↑**	**24% ↑ 12X ↑**	**15% ↑ 50X ↑**	**16% ↑ 177X ↑**	**37% ↑ 1X ↓**
TTM_A *f-imp(%) s-imp(X)*				**28% ↑ 2X ↑**	**22% ↑ 9X ↑**	**12% ↑ 40X ↑**	**13% ↑ 142X ↑**	**37% ↑ 2X ↓**

Table 2: **Zero-shot** forecast-improvement *(f-imp)* and model size-improvement *(s-imp)* of TTM over Chronos and Lag-llama over the last test-window. Since Chronos and Lag-llama recommend/report results with shorter forecast lengths, we use $FL \in \{24, 48, 60, 96, 192\}$. Mean MSE across FLs is reported. Full table in the Appendix F.2

Model	GPU TIME (ms)	Params (M)	MEM (GB)	CPU TIME (s)
TTM_B	**4.7**	**0.8**	**0.06**	**0.01**
Chronos_B (2024)	1395 (298X)	201 (251X)	16 (267X)	2340 (239KX)
Chronos_L (2024)	1393 (298X)	709 (886X)	41 (683X)	2352 (240KX)
Chronos_S (2024)	1386 (296X)	46 (58X)	6 (100X)	2349 (240KX)
Chronos_T (2024)	1389 (297X)	8 (10X)	2 (33X)	2504 (256KX)
GPT4TS (NeurIPS '23)	13.9 (3X)	87 (109X)	1.34 (36X)	0.3 (26X)
Lag-Llama (2024)	1619 (346X)	2.4 (3X)	0.2 (3X)	37.5 (3830X)
Moirai_S (ICML '24)	205 (44X)	14 (18X)	0.1 (2X)	1.4 (141X)
Moirai_L (ICML '24)	693 (148X)	311 (389X)	2 (33X)	10.5 (1070X)
Moirai_B (ICML '24)	335 (72X)	91 (114X)	1 (17X)	4.1 (421X)
Moment-L (ICML '24)	88 (19X)	348 (435X)	8 (133X)	1.4 (144X)
TimesFM (ICML '24)	24 (5X)	200 (250X)	2 (33X)	0.4 (46X)

Table 3: **Computational improvement** of TTM *w.r.t.* existing TS pre-trained models. Inference time per-batch in GPU and CPU, total parameters (Params), and maximum GPU memory usage (MEM) are reported. nX indicates the scaling factor for TTM's improvement. Set-up details are in the Appendix D.3

4.4 TTM's Zero-shot Performance and Inference Cost: Recently, popular pre-trained models like TimesFM, Moirai, Chornos, Lag-llama, and LLMTime have gained traction for their zero-shot (ZS) forecasting capabilities. Among these, Chornos, Lag-llama, and LLMTime suffer from lengthy ZS inference time, posing practical challenges for testing across all sliding windows of the test set. To address this, LLMTime suggests using the last test window for benchmarking, a practice we also adopt for comparing with this set of SOTA models. On the other hand, TimesFM and Moirai exhibit comparatively faster ZS inference speeds, enabling testing across all sliding windows of the test set. Table 1 presents a comparison of TTM performance with Moirai and TimesFM. Despite having significantly fewer parameters, the variants of TTM consistently outperform most benchmark variants. Notably, TTM_A, which is 3-62X smaller than all Moirai variants and 40X smaller than TimesFM, outperforms the Moirai variants by 4-10% and TimesFM by 19%. Even TTM_B, with just 1M parameters, outperforms most benchmarks by a considerable margin, highlighting the effectiveness of TTM. Moreover, as depicted in Appendix F.4, TTM zero-shot results consistently outperform the full-shot results of popular architectures in short context length settings. Likewise, Table 2 presents a comparison of TTM performance with Chronos and Lag-llama on the last test-window set. As indicated, TTM_B which is 8-709X smaller than Chronos, outperforms it by 17-32%. Likewise TTM_B, which is 2-3X smaller than Lag-llama, outperforms it by 40%. In addition, TTM also outperforms the massive LLMTime and UniTime by over 25% as reported in Appendix F.3. Table 3 presents the inference time per batch and maximum GPU memory requirement of different TS pre-trained models. Notably, TTM exhibits the lowest inference time and memory usage among them.

4.5 TTM's Few-shot and Full-shot Head Probing Performance: In operational deployments, users typically leverage a small set of target data for fine-tuning to enhance the model performance. In this regard, TTM provides a highly efficient quick fine-tuning process, enabling users to enhance forecasting accuracy swiftly by training only the model head. GPT4TS and Time-LLM are two state-of-the-art pre-trained models that present results for few-shot training. As demonstrated in

Table 4, TTM_B surpasses GPT4TS by 15% and Time-LLM by 10% in the few-shot 5% setting, where only 5% of the train data is used for fine-tuning. In addition, we also report the Few-shot 5% results of several popular SOTA architectures in Table 4, where TTM demonstrates superior performance. This underscores the significance of TTM's pre-trained weights, which substantially contribute to its effectiveness in data-constrained scenarios. Likewise, TTM also excels in few-shot cross-transfer learning tasks outperforming popular SOTAs (including SimMTM [5]) as shown in the Appendix F.6.

Alternatively, if the train split of the complete target dataset is available, head probing using the entire dataset becomes feasible. This involves fine-tuning the model head using all available data while keeping the backbone weights unchanged. Recently, the Moment [10] model has achieved the SOTA results in head probing as compared to GPT4TS and Time-LLM. However, as indicated in Table 5, TTM further outperforms the results reported by Moment by 3-4%. In addition, TTM head probing results are very competitive as compared to the full end-to-end training of popular architectures as depicted in Appendix F.7. Hence, TTM, with its significantly reduced model size and the absence of compute-intensive components like self-attention, enables quick fine-tuning of models compared to the cumbersome process required by the massive Transformer models. Note that Moment is excluded from the comparison of zero/few-shot forecasting results because it does not report them.

	Pre-trained models fine-tuned on 5% data					**Other popular architectures trained on 5% data**					
Data	**TTM_B** (Ours)	**TTM_E** (Ours)	**TTM_A** (Ours)	**GPT4TS** (NeurIPS '23)	**Time-LLM** (ICLR '24)	PatchTST (ICLR '23)	TSMixer (KDD '23)	TimeMixer (ICLR '24)	iTransformer (ICLR '24)	TimesNet (ICLR '23)	Dlinear (AAAI '23)
ETTH1	**0.383**	0.385	0.386	0.682	0.627	0.695	0.635	1.088	0.756	0.926	0.75
ETTH2	0.324	0.318	**0.314**	0.401	0.382	0.439	0.385	0.508	0.437	0.464	0.828
ETTM1	0.376	0.378	**0.361**	0.472	0.425	0.526	0.479	0.578	0.568	0.717	0.401
ETTM2	0.272	0.268	**0.253**	0.308	0.274	0.314	0.297	0.34	0.309	0.344	0.399
Weather	0.234	0.24	**0.229**	0.263	0.261	0.27	0.268	0.317	0.297	0.298	0.264
Electricity	0.183	0.207	0.18	0.178	0.177	**0.176**	0.197	0.239	0.202	0.402	0.177
Traffic	0.433	0.437	0.49	0.434	0.423	**0.418**	0.435	0.503	0.452	0.867	0.451
Size	**1M**	**4M**	**5M**	**84M**	**7B**						
TTM_B *f-imp(%) s-imp(X)*				**15% ↑ 84X ↑**	**10% ↑ 7KX ↑**	**17% ↑**	**15% ↑**	**31% ↑**	**22% ↑**	**40% ↑**	**23% ↑**
TTM_E *f-imp(%) s-imp(X)*				**13% ↑ 21X ↑**	**8% ↑ 1.7KX ↑**	**15% ↑**	**13% ↑**	**30% ↑**	**20% ↑**	**40% ↑**	**21% ↑**
TTM_A *f-imp(%) s-imp(X)*				**15% ↑ 17X ↑**	**11% ↑ 1.4KX ↑**	**17% ↑**	**15% ↑**	**32% ↑**	**22% ↑**	**41% ↑**	**23% ↑**

Table 4: **Few-shot 5%**. MSE averaged across $FL \in \{96, 192, 336, 720\}$, models are trained with 5% train data (Appendix F.5).

Data	TTM_B (Ours)	TTM_E (Ours)	TTM_A (Ours)	Moment (ICML '24)	GPT4TS (NeurIPS '23)	Time-LLM (ICLR '24)
ETTH1	**0.398**	0.406	0.402	0.42	0.426	0.466
ETTH2	0.33	0.338	**0.327**	0.346	0.346	0.342
ETTM1	0.355	0.35	**0.338**	0.349	0.354	0.41
ETTM2	0.257	**0.252**	0.264	0.266	0.275	0.273
Weather	0.234	0.239	**0.233**	0.234	0.244	-
Electricity	0.164	0.161	**0.16**	0.174	0.172	-
Traffic	0.399	0.398	**0.385**	0.42	0.419	-
Size	**1M**	**4M**	**5M**	**348M**	**84M**	**7B**
TTM_B *f-imp(%) s-imp(X)*				**3% ↑ 348X ↑**	**4% ↑ 84X ↑**	**9% ↑ 7000X ↑**
TTM_E *f-imp(%) s-imp(X)*				**3% ↑ 87X ↑**	**4% ↑ 21X ↑**	**9% ↑ 1750X ↑**
TTM_A *f-imp(%) s-imp(X)*				**4% ↑ 70X ↑**	**6% ↑ 17X ↑**	**10% ↑ 1400X ↑**

Table 5: **Full-shot head probing:** Finetuning the pre-trained model heads on full data with backbone weights frozen. MSE averaged across *FL* 96, 720 as reported in [10]. Time-LLM results for large datasets are not reported in [10] due to computational challenges (AppendixF.7).

Models	BS	CC	APP	SER	*f-imp%*
TTM_Q	0.635	0.261	0.073	0.143	18%
PatchTST	0.735	0.267	0.060	0.119	15%
TSMixer-CC	0.651	0.284	0.053	0.136	15%
TSMixer-CM	0.716	0.303	0.069	0.118	20%
TSMixer	0.664	0.267	0.066	0.134	17%
GPT4TS	0.645	0.254	0.075	0.135	18%
TTM_Q-CM	**0.582**	**0.250**	**0.042**	**0.114**	

Table 6: **Effect of decoder mixing and exog. fusion.** MSE results are reported using (sl, fl) with values of $(512, 96)$ for BS dataset and $(96, 24)$ for other D2 datasets. *f-imp%* of TTM_Q-CM *w.r.t.* others are provided.

4.6 TTM's Effectiveness in Cross-channel and Exogenous Modeling: Since the datasets (**D1**) used in previous experiments do not have exogenous variables, we evaluate the effectiveness of TTM on 4 other datasets (**D2**, as explained in Section 4.1) to quantify its benefits. Since these datasets are already very small, we used their full data for fine-tuning. Table 6 shows the performance of the pre-trained TTM_Q model fine-tuned on the target data with exogenous mixer module and decoder channel-mixing enabled (TTM-CM). We compare TTM-CM with plain TTM finetuning and other primary SOTAs (PatchTST, TSMixer variants, and GPT4TS) trained from scratch. Specifically, we compare with TSMixer with channel-mixing enabled (TSMixer-CM) and TSMixer with cross-channel reconciliation head (TSMixer-CC) [6] as they are the latest SOTAs in channel-correlation modelling. From Table 6, we can see that TTM-CM outperforms all the competitive models with a significant margin (15-44%), thus, demonstrating the power of TTM in capturing inter-channel correlations.

4.7 Ablation Studies: The impacts of various techniques used in TTM are analyzed here.
Pre-training data (Quality Vs Quantity): Figure 3 demonstrates the vital role of both pretraining

data and diverse resolution sampling (DRS). Initially, the zero-shot results were unsatisfactory when pre-training TTM with the smaller Monash dataset (i.e., PT(M)). To improve performance, we introduced the DRS technique on the Monash data to increase diversity and coverage (250M PT samples). This significantly improved the results by 37%. In addition, extending the dataset size from 250M to 1B further improved the results by 6%. These experiments highlight that while the quantity of pre-training data is significant, the quality of the data, especially in terms of resolution diversity and coverage, is even more crucial for improving the model performance.

Effect of Resolution Prefix Tuning (RPT) and Adaptive Patching(AP): RPT enhances forecast performance, especially with large and diverse pretraining (PT) data. Adding a learnable resolution prefix token allows models to easily decouple weights across different resolutions, leading to a 3% improvement in 1B PT data setup (Table 7). RPT is also beneficial for very short context length scenarios, improving performance by 8% (Appendix F.9). On the other hand, AP generally improves the forecasting performance across all set-ups, but the impact is consistently high in less PT data settings (3% boost). Further details are in Appendix F.8.

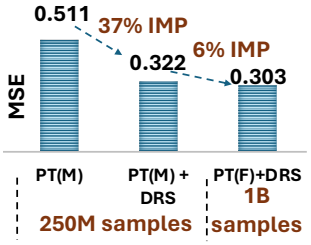

Figure 3: **Impact of pre-training data (PT) and diverse resolution sampling (DRS) technique.** PT(M): pretraining with only Monash data. PT(F): full pretraining data used. Average MSE of zero-shot results across *FL* 96, 192 reported.

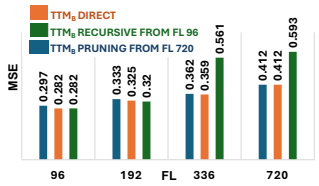

Figure 4: **FL adaptation:** impact of adapting TTM_B (*FL* 720) and TTM_B (*FL* 96) to all other FLs. MSE averaged across all D1 datasets is reported for $FL \in \{96, 192, 336, 720\}$. Best viewed in color.

	Less PT Data (250M)		More PT Data (1B)	
Data	w/o AP	w/ AP	w/o RPT	w/ RPT
ETTH1	0.369	**0.365**	0.366	**0.364**
ETTH2	**0.283**	0.285	0.285	**0.277**
ETTM1	0.446	**0.413**	0.341	**0.322**
ETTM2	0.191	**0.187**	0.18	**0.171**
Weather	0.159	**0.154**	**0.153**	0.158
Electricity	0.179	**0.169**	0.178	**0.166**
Traffic	0.521	**0.518**	0.528	**0.514**
IMP (%)	**3%**		**3%**	

Table 7: **Impact of AP and RPT:** Impacts of adaptive patching (AP) in less pre-training (PT) data setting (i.e., TTM_Q), and resolution prefix tuning (RPT) in more pre-training (PT) data setting (i.e., TTM_B). Zero-shot results on FL 96 reported. ['w/': with, 'w/o': 'without'.]

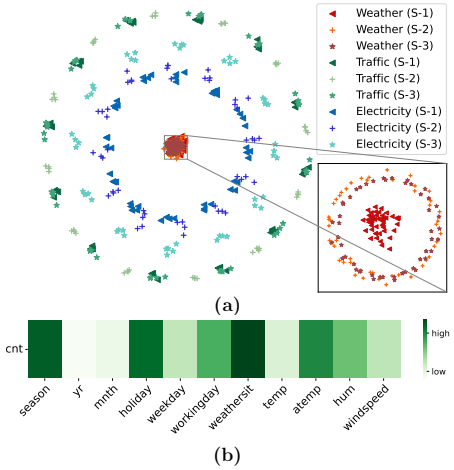

Figure 5: (a) **TTM embedding projections** across 3 datasets and 3 segments within datasets. (b) Cross-channel **attention based explanation**.

Forecast Length Adaptations (FLA): Given a *FL*, we can either pre-train a Direct TTM tailored for the specific *FL* or adapt existing TTMs trained on different *FL*s. Two possible adaptations are: (i) **Pruning:** Take TTM trained on FL' where $FL' > FL$, and prune it to the required *FL* (e.g., TTM $(FL = 720)$ pruned to other reduced $FL \in \{96, 192, 336\}$). (ii) **Recursive:** Take TTM trained on FL', where $FL' < FL$ and do recursive prediction (of length FL') till we reach the required *FL* (e.g., Extend TTM $(FL = 96)$ recursively to greater $FL \in \{192, 336, 720\}$). Figure 4 compares these techniques. For shorter adaptation (96 to 192), recursive predictions yield the best performance and match the direct forecast results. However, for wider adaptations (336-96 or 720-96), the pruning approach gives more stable and closer results to the direct forecasts. Hence, using these approaches, TTM models can be easily adapted to various *FL*s based on user requirements.

4.8 TTM Model Insights & Explainability: Figure 5 illustrates the TTM embeddings from various datasets (weather, traffic, and electricity) using PCA projection, each represented by a different color. From each dataset, three distant, non-overlapping, fixed-length time segments (**S-1**, **S-2**, **S-3**) are selected, each depicted with a unique marker shape. The visualization uses the first and second principal components of the TTM embeddings. The inset image focuses on the weather dataset alone, revealing a deeper structure learned by the TTM architecture. The cyclic orbits in the embeddings reflect the seasonal patterns in the data. Both

hourly datasets (traffic and electricity) form concentric orbits due to similar seasonal patterns, while the weather data, with its distinct seasonal pattern, shows cyclic orbits in a different sub-dimension. In addition, the cross-channel attention from the fine-tuned model's channel mixing layers reveals feature importance across channels. As shown in Figure 5, the model focuses on channels like weathersit, season, holiday, and temperature to predict bike-rental counts. These attention model weights correlate with the general data characteristics where bike rental demands are heavily influenced by weather and holidays, providing explanation for the fine-tuned model predictions. More details are in Appendix G.

4.9 Discussion on TTM Design choices: In this section, we intuitively explain the important design choices of TTM that greatly enhance its forecasting accuracy and transfer learning capabilities despite its extremely small model capacity:

- All existing pre-trained models use a very high volume of pretraining data (for example, TimesFM used 300B and Moirai used 27B time-points), hence they naturally require massive model sizes. However, as shown in Figure.3, we observe that "limited" pretraining data with "high resolution diversity" greatly helps in time-series model generalization, as opposed to simply increasing the pretraining data size. This is an important observation and finding that resolution diversity in pretraining data is very crucial for time-series FMs. Based on these findings, we proceed with a well-reduced dataset (1B samples) with high resolution diversity which naturally reduces our model size compared to counterparts needing to pretrain with several hundred billion time-series. We introduce a high diversity in our data via Diverse Resolution Sampling technique (DRS) which our counterparts fail to do.
- Secondly, we opted for TSMixer-based models instead of transformer-based models, which further reduced the model size drastically. The TSMixer architecture has successfully established in the past that interleaving simple gated attentions with mixing components across patches, channels, and features greatly enhances forecasting accuracies with very limited model capacity, as the quadratic time-complexity of self-attentions can be entirely avoided. Following TSMixer, several other mixer architectures [33] [24] have been published, reiterating the power of these simple architectures. Thus, avoiding complex transformer architectures further reduced our model size significantly.
- In addition, we further increased the modeling power of TSMixer without drastically increasing its size by introducing several innovative components, such as adaptive patching, diverse resolution sampling, and resolution prefix tuning. These enhancements are crucial for effectively handling large pre-training across datasets with varying resolutions, all while keeping the model capacity very minimal.
- Finally, framing the pre-training objective as a direct forecasting task demonstrates improved zero-shot performance as compared to the traditional masking-based pre-training approaches. We hypothesize that this method enables the model to effectively learn complex non-linear mappings between the fixed context and forecast windows during pre-training that generalizes well to unseen datasets.

5 Conclusions and Future Work

We propose TTM, an extremely lightweight pre-trained model for multivariate time-series forecasting. Unlike existing large models, TTM is significantly smaller and faster, with efficient pre-training and fine-tuning workflows. Results show that TTM is highly effective in pre-training on heterogeneous datasets despite its limited model capacity. It achieves state-of-the-art results in zero/few-shot forecasting, offering significant computational efficiency while capturing cross-channel relationships and exogenous variables – critical features lacking in popular methods. Additionally, TTM supports both CPU and GPU deployments, greatly enhancing its adoption and ease of use. Moving forward, we plan to generalize our approach to support other downstream tasks beyond forecasting.

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical re-

port: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] 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.

[3] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. *International Conference on Machine Learning (ICML)*, 2023.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[5] Jiaxiang Dong, Haixu Wu, Haoran Zhang, Li Zhang, Jianmin Wang, and Mingsheng Long. Simmtm: A simple pre-training framework for masked time-series modeling. In *Advances in Neural Information Processing Systems*, 2023.

[6] Vijay Ekambaram, Arindam Jati, Nam Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. Tsmixer: Lightweight mlp-mixer model for multivariate time series forecasting. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 459–469, New York, NY, USA, 2023.

[7] Hadi Fanaee-T. Bike Sharing Dataset. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C5W894.

[8] Azul Garza and Max Mergenthaler-Canseco. Timegpt-1, 2023.

[9] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, Rob J. Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. In *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[10] Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. *International Conference on Machine Learning (ICML)*, 2024.

[11] Nate Gruver, Marc Anton Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[12] R.J. Hyndman and G. Athanasopoulos, editors. *Forecasting: principles and practice*. OTexts: Melbourne, Australia, 2021. `OTexts.com/fpp3`.

[13] Kevin Maik Jablonka, Charithea Charalambous, Eva Sanchez Fernandez, Georg Wiechers, Juliana Monteiro, Peter Moser, Berend Smit, and Susana Garcia. Machine learning for industrial processes: Forecasting amine emissions from a carbon capture plant. *Science Advances*, 9(1):eadc9576, 2023.

[14] Arindam Jati, Vijay Ekambaram, Shaonli Pal, Brian Quanz, Wesley M. Gifford, Pavithra Harsha, Stuart Siegel, Sumanta Mukherjee, and Chandra Narayanaswami. Hierarchical proxy modeling for improved hpo in time series forecasting. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 891–900, New York, NY, USA, 2023.

[15] Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-LLM: Time series forecasting by reprogramming large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[16] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021.

[17] Zhe Li, Zhongwen Rao, Lujia Pan, Pengyun Wang, and Zenglin Xu. Ti-mae: Self-supervised masked time series autoencoders. *arXiv preprint arXiv:2301.08871*, 2023.

[18] Xu Liu, Junfeng Hu, Yuan Li, Shizhe Diao, Yuxuan Liang, Bryan Hooi, and Roger Zimmermann. Unitime: A language-empowered unified model for cross-domain time series forecasting. In *Proceedings of the ACM Web Conference 2024*, 2024.

[19] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting, 2024.

[20] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[21] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 2022. `https://doi.org/10.1016/j.ijforecast.2021.11.013`.

[22] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *ICLR*, 2023.

[23] 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*, 2020.

[24] Santosh Palaskar, Vijay Ekambaram, Arindam Jati, Neelamadhav Gantayat, Avirup Saha, Seema Nagar, Nam H Nguyen, Pankaj Dayama, Renuka Sindhgatta, Prateeti Mohapatra, et al. Automixer for improved multivariate time-series forecasting on business and it observability data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 22962–22968, 2024.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[26] Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Arian Khorasani, George Adamopoulos, Rishika Bhagwatkar, Marin Biloš, Hena Ghonia, Nadhir Vincent Hassen, Anderson Schneider, et al. Lag-llama: Towards foundation models for time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

[27] BizITOps Dataset Repository. `https://github.com/BizITObs/BizITObservabilityData/tree/main/Complete/Time%20Series/RobotShop`, 2023.

[28] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[29] Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*, 2020.

[30] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[32] Jingyuan Wang, Jiawei Jiang, Wenjun Jiang, Chao Li, and Wayne Xin Zhao. Libcity: An open library for traffic prediction. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '21, page 145–148, New York, NY, USA, 2021.

[33] Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y Zhang, and JUN ZHOU. Timemixer: Decomposable multiscale mixing for time series forecasting. In *International Conference on Learning Representations (ICLR)*, 2024.

[34] Zhiyuan Wang, Xovee Xu, Weifeng Zhang, Goce Trajcevski, Ting Zhong, and Fan Zhou. Learning latent seasonal-trend representations for time series forecasting. *Advances in Neural Information Processing Systems*, 35:38775–38787, 2022.

[35] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. *International Conference on Machine Learning (ICML)*, 2024.

[36] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. CoST: Contrastive learning of disentangled seasonal-trend representations for time series forecasting. In *International Conference on Learning Representations*, 2022.

[37] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The Eleventh International Conference on Learning Representations*, 2022.

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

[39] Zhengqing Yuan, Zhaoxu Li, and Lichao Sun. Tinygpt-v: Efficient multimodal large language model via small backbones. *arXiv preprint arXiv:2312.16862*, 2023.

[40] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8980–8987, 2022.

[41] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *arXiv preprint arXiv:2205.13504*, 2022.

[42] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2114–2124, 2021.

[43] Xiang Zhang, Ziyuan Zhao, Theodoros Tsiligkaridis, and Marinka Zitnik. Self-supervised contrastive pre-training for time series via time-frequency consistency. *Advances in Neural Information Processing Systems*, 35:3988–4003, 2022.

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

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

[46] Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. One Fits All: Power general time series analysis by pretrained lm. In *NeurIPS*, 2023.

Appendix

A TSMixer Background

We employed TSMixer [6] as a building block for the proposed TTM model due to its state-of-the-art performance, faster execution, and significantly lower memory usage. However, as explained in the main paper, vanilla TSMixer cannot be trained on multiple diverse datasets. Therefore, it necessitated the incorporation of the proposed novel components. In this section, we provide a high-level overview of the TSMixer model for a simpler and quicker understanding by the readers.

TSMixer is a lightweight alternative to transformer-based time series models, with no compromise on forecast accuracy. TSMixer adopts some well-established pre-processing steps from the literature, such as normalization and patching. Additionally, it offers the flexibility of enabling or disabling channel mixing. Channel mixing has been found to be beneficial in handling multivariate datasets with cross-channel correlations. For the main learning process, TSMixer employs a series of MLP-Mixer [30] blocks that perform inter-patch, intra-patch, and inter-channel mixing operations. A mixing operation in TSMixer ensures learning correlations across a specific dimension. For example, inter-channel mixing enables it to learn cross-channel correlations. In the experiments, we employed three different flavors of the TSMixer model: TSMixer vanilla (referred as TSMixer throughout the text), TSMixer with cross-channel mixing enabled (TSMixer-CM), and TSMixer with cross-channel reconciliation head (TSMixer-CC). We request the readers to refer to [6] for further details about these variants.

B Literature Survey

B.1 Multivariate Time Series Forecasting: Statistical approaches for time series forecasting, such as SARIMAX and Exponential Smoothing, generally generate forecasts independently for each time series [12]. These methods are essentially univariate and do not build a single model by learning from multiple time series. On the other hand, more advanced models, built upon machine/deep learning techniques, including LightGBM-based models [21, 14], N-BEATS [23], and DeepAR [28], have the capability to learn from multiple time series. However, these models still follow univariate approaches, thus ignoring any potential cross-channel correlations.

Advanced multivariate forecasting models mostly involve deep neural networks, specifically the transformer [31] architecture. A series of transformer-based model have been proposed in the last few years including Informer [44], Autoformer [38], and FEDFormer [45]. Although these models outperformed all the prior arts, the DLinear [41] model showed that an embarrassingly simple linear model can beat these models by following a few empirically established steps like time series decomposition, normalization, and channel-independent modeling.

PatchTST [22] showed that transformers can be effective for forecasting if the input time series is patched or segregated in multiple windows, and subsequently, modeled by a transformer. The patching operation helps preserve local semantic information, accommodates a longer history, and reduces computation time. The PatchTST model outperformed all prior transformer-based models and the DLinear model.

Although PatchTST reinstated faith in transformers for time series modeling, transformer-based models are generally resource-intensive, with slow execution and a high memory footprint. The recently proposed TSMixer model [6] addresses these challenges effectively. TSMixer, built on the MLPMixer architecture [30], stands out for its exceptional speed and lightweight design. It has attained state-of-the-art (SOTA) performance on benchmark datasets, demonstrating a 2-3X reduction in both execution time and memory usage.

Recently, several new Transformer- and Mixer-based architectures have been proposed. The iTransformer model [19] applies attention and MLP modules to the inverted dimension. Instead of operating on the temporal tokens, these operations are applied to the variate tokens, resulting in "variate-unmixed representations". This approach is claimed to enhance generalization across different channels and improve the use of arbitrary context lengths. The TimeMixer model [33] leverages the observation that time series exhibit unique patterns at different sampling scales. By utilizing

different MLPMixer blocks, it aims to capture both microscopic and macroscopic information to produce more accurate forecasts. Similarly, the recent TimesNet model [37] disentangles the complex multi-periodicity in a time series into intra-period and inter-period variations. It then learns time series representations using an Inception block, enhancing the model's ability to capture intricate patterns in the data.

B.2 Pre-trained Models for Time Series: One major drawback of all the above models is that they need to be trained in-domain. Hence, none of these models can be transferred to out-of-domain data with zero or minimal training. This approach has been found to be extremely beneficial in the natural language processing (NLP) domain with the invention of BERT [4] and GPT [25] models.

However, this is an extremely challenging task in the time series domain because of the unavailability of a publicly accessible large pre-training corpora. There are multiple independent time series datasets, but, unlike in NLP, these datasets differ significantly in important characteristics such as the domain of the data (e.g., retail, sensor data, traffic, etc.), the number of channels, temporal resolution, and length. This makes it hard to train a single model on all the datasets together.

Hence, a few prior works have focused on experimenting with *same-dataset* self-supervised learning for time series [17, 34, 36, 40]. These methods learn a time series representation from the *train* split of a dataset, build a forecaster on top of the learned representation on the same data, and then evaluate it on the *test* split of the same dataset. Although these approaches have demonstrated promising results, they do not provide evidence of the transfer capability of the model between datasets.

Subsequently, models such as SimMTM [5] and TF-C [43] have demonstrated the transfer capabilities of their models between pairs of datasets. These pairs are carefully chosen so that the *source* (the dataset where the model is pre-trained) and *target* (the dataset where the model is fine-tuned and tested) datasets share some matching properties. For instance, SimMTM showcased its few-shot capability by selecting ETTH2 as the source data and ETTH1 as the target data. Both ETTH1 and ETTH2 are collected from Electricity Transformers at two stations, denoting data from a similar domain. TF-C demonstrated the transferability of the model across four different (source, target) pairs, such as (ECG, EMG) and (FD-A, FD-B), where domain-similarity exists in both the source and target datasets.

To overcome this limitation, the time series research community is increasingly focused on developing General Pre-Trained (GPT) or Foundation Models (FM) for time-series forecasting, capable of effectively transferring knowledge to new target TS datasets. This growing interest led to the release of several "large" and "massive" pre-trained time-series models for forecasting in early 2024, generating significant excitement among researchers. Notable releases include Moment [10], TimesFM [3], Chronos [2], Moirai [35], and Lag-llama [26], all of which set strong benchmarks in zero-shot forecasting. The Moment [10] model pre-trains a Transformer encoder model in a univariate way on a collected set of diverse "Time Series Pile". Moment is pre-trained with mask reconstruction objective, and it can be fine-tuned on a downstream forecasting task. The TimesFM [3] pre-trains a decoder-style attention model (with causal self-attention) in univariate fashion on a large collection of real world and synthetic datasets. The Chronos [2] model tokenizes the input time series, and feed the tokens into a large langugae model (specifically the T5 model). Chronos is pre-trained in a univariate fashion. During inference, Chronos auto-regressively samples tokens and map them to the numerical values via dequantization. Chronos is trained on a large corpora of time series including synthetic data for better generalization. The Moirai [35] model pre-trains a Transformer encoder on a massive collection of "LOTSA" dataset (27B time points). Moirai masks the forecast horizon of each target channel and performs mask reconstruction. The flattening operation of all channels in a multivariate time series enables Moirai to pre-train on "any-variate" settings. The Lag-Llama [26] model pre-trains a decoder-only Transformer model that utilizs the time series lags as covariates. Lag-Llama is pre-trained on a large collection of diverse time series datasets in a univariate fashion. All the above models are open-sourced and used in our experiments for comparison. However, closed-source models such as TimeGPT [8] are not included due to their inaccessibility.

B.3 Pre-trained LLMs for Time Series: Parallel to the above trend of general pre-trained TS models, there has been a notable increase in the adoption of pre-trained large language models (LLMs) for time series tasks. These models are approached as cross-domain transfer learning problems. The LLMTime model [11] feeds the time series values as text representations and demonstrates promising

Source	Dataset	Resolution
Monash	kaggle_web_traffic_dataset_without_missing_values	daily
	nn5_daily_dataset_without_missing_values	daily
	solar_10_minutes_dataset + **Downsample**	10 mins, 30 mins, hourly
	australian_electricity_demand_dataset + **Downsample**	30 mins, hourly, daily
	solar_4_seconds_dataset + **Downsample**	4 seconds, 10 mins, 15 mins, 30 mins, hourly
	wind_4_seconds_dataset + **Downsample**	4 seconds, 10 mins, 15 mins, 30 mins, hourly
	us_births_dataset	daily
	saugeenday_dataset	daily
	sunspot_dataset_without_missing_values	daily
	australian_weather_dataset	daily
	kdd_cup_2018_dataset_without_missing_values	hourly
	bitcoin_dataset_without_missing_values	daily
	wind_farms_minutely_dataset_without_missing_values + **Downsample**	minutely, 10 mins, 15 mins, 30 mins, hourly
	london_smart_meters_dataset_without_missing_values + **Downsample**	30 mins, hourly, daily
LibCity	PEMS03 + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	PEMS04 + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	PEMS07 + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	PEMS08 + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	PEMS_BAY + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	LOS_LOOP + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	LOOP_SEATTLE + **Downsample**	5 mins, 10 mins, 15 mins, 30 mins, hourly
	SZ_TAXI + **Downsample**	15 mins, 30 mins
	Q-TRAFFIC + **Downsample**	15 mins, 30 mins, hourly

Table 8: List of pre-training datasets. A dataset with "**+ Downsample**" denotes that the proposed Diversity Resolution Sampling (DRS) has been applied on that dataset to generate new diverse datasets at frequencies lower than the original frequency of the data. Please note that, these pre-training datasets have no overlap with the evaluation datasets. Specifically, the australian_electricity_demand_dataset and australian_weather_dataset used in pre-training are completely different (*w.r.t* location, measured variables, type, resolution, length, etc.) from the standard Electricity (ECL) and Weather dataset used in the evaluation. Please note that, the last three datasets in the Libcity section have been excluded from the pre-training process for the model releases intended for enterprise-use.

performance in a zero-shot setting. The GPT4TS model [46] adopts a pre-trained LLM like GPT and fine-tunes only the input embedding layer, normalization layers, and output layer. Specifically, it does not alter the self-attention weights and feed-forward layers. The Time-LLM [15] model proposed a reprogramming framework, where they reuse existing LLMs for forecasting while keeping the LLM backbone intact. The overall approach to building a pre-trained model for time series from LLMs is promising, but it does not model cross-channel correlations observed in many multivariate time series datasets. Moreover, these LLMs are very large and exhibit slow execution and a large memory footprint.

C Datasets

C.1 List of Pre-training Datasets: Pre-training employs a subset of $\sim$1B samples from Monash [9] and Libcity [32, 35] data collection, where Monash results in $\sim$250M samples and LibCity accounts for the rest. In this estimate, one sample denotes a pair of training windows: $X \in \mathbb{R}^{1\times sl}$ and $Y \in \mathbb{R}^{1\times fl}$. We employ a subset of the datasets available in the Monash forecasting data repository [9] available at `https://forecastingdata.org/`. Since our primary focus in this study is long term forecasting with forecast length ranging from 96 to 720, it is not possible to use yearly, monthly, quarterly, or weekly datasets due to their short lengths. Hence, we skip a few datasets of short lengths. The Monash datasets used are available under a Creative Commons Attribution 4.0 International license. For LibCity, we employ all datasets released by the Moirai authors [35], available at `https://huggingface.co/datasets/Salesforce/lotsa_data/tree/main` (except the Rotterdam dataset which was not available during our experimentation). The LibCity datasets at the above link were released under an Apache 2.0 license. The final list of all pre-training datasets is shown in Table 8. Please note that, the last three datasets in the Libcity section have been excluded from the pre-training process for the model releases intended for enterprise-use.

C.1.1 Temporal cross-validation: Temporal cross-validation is used to chronologically split all the time series into train and validation parts. During pre-training, moving windowing technique is used to create $(\boldsymbol{X}, \boldsymbol{Y})$ pairs of lengths sl and fl respectively. Please note that, these pre-training datasets have no overlap with the evaluation datasets. Specifically, the aus-

Set	Dataset	Resolution	Length	Total #Channels	#Target variables	#Exog. variables	Source
D1	ETTH1	1 hour	17420	7	7	Not Applicable	[38]
	ETTH2	1 hour	17420				
	ETTM1	15 minutes	69680				
	ETTM2	15 minutes	69680				
	Weather	10 minutes	52696	21	21		[38]
	ECL	1 hour	26304	321	321		[38]
	Traffic	1 hour	17544	862	862		[38]
D2	BS	1 hour	17379	14	3	11	[7]
	CC	2 minutes	5409	8	2	5	[13]
	APP	10 seconds	8834	39	4	35	[27]
	SER	10 seconds	8835	107	72	35	[27]

Table 9: Details of the evaluation datasets.

tralian_electricity_demand_dataset and australian_weather_dataset used in pre-training are completely different (*w.r.t* location, measured variables, type, resolution, length, etc.) from the standard Electricity (ECL) and Weather dataset used in the evaluation.

C.2 List of Evaluation Datasets: Table 9 illustrates various characteristics of the eleven evaluation datasets. Below, we present the details.

Set D1: For zero/few/full-shot evaluation, we utilize seven multivariate time series datasets that have consistently been employed in the literature. Below, we offer a brief overview of these datasets.

1. **ETT datasets:** The four ETT datasets [44] (ETTH1, ETTH2, ETTM1, ETTM2) contain multivariate time series data collected from electrical transformers at two stations. ETTH1 and ETTH2 are collected at an hourly interval, while ETTM1 and ETTM2 are collected every 15 minutes. All four datasets have 7 channels.
2. **Weather:** The weather dataset consists of 21 channels, which serve as weather indicators. It is collected at 10-minute intervals at the Max Planck Institute of Biogeochemistry weather station.
3. **Electricity (ECL):** The Electricity dataset, also known as the ECL dataset, comprises the hourly electricity consumption data of 321 clients.
4. **Traffic:** This dataset records the hourly rates of road occupancy on the San Francisco Freeways using 862 sensors.

We used the datasets provided in the repository of the Autoformer paper [38][7]. For all the D1 datasets, we execute the same train/validation/test splitting as was performed in the literature [44, 38, 22, 6].

Set D2: To assess the effectiveness of the proposed TTM model in extracting information from exogenous channels, we conduct evaluations on four additional datasets that are known to contain exogenous or control variables.

1. **Bike Sharing (BS):** The Bike Sharing dataset [7] documents the hourly rental counts of bikes from the Capital Bikeshare system in Washington D.C., USA, spanning the years 2011 to 2012. Rental counts are typically associated with environmental and seasonal conditions. Consequently, this 14-channel dataset encompasses various weather-related features. Our goal was to forecast all three rental counts: "casual", "registered", and "cnt" (total count). As the remaining 11 features are consistently available at all future time points, they are treated as exogenous variables in our experiment.
2. **Carbon Capture Plant (CC):** The Carbon Capture Plant data [13] records the emission profiles of "2-amino-2-methyl-1-propanol" (AMP) and "piperazine" (Pz) collected at every 2 minutes interval. We utilize the 8-channel dataset made available in the official repository [13]. Among the remaining 6 channels, the following 5 serve as control variables:

[7]Available at `https://github.com/thuml/Autoformer`

["TI-19","FI-19", "TI-3", "FI-11", "TI-1213"]. The remaining 1 variable is treated as a conditional variable (as it is neither a target variable nor available during the forecast period to consider it as exogenous). For additional details, please refer to the supplementary materials of [13].

3. **Service (SER):** This dataset pertains to the cloud-based "Stan's Robot Shop" application, managed by Instana. It simulates a user's e-commerce experience, encompassing site access to shipping, utilizing a load generator. Intermittent fault injection introduces diverse IT events. The dataset provides business KPIs for services (e.g., payment, catalog) and IT events tracked by Instana. Sampling occurs every 10 seconds due to high traffic and event frequency. For our experiments, all business KPIs are treated as target variables and IT events are treated as exogenous variables and the goal of our forecasting is to predict the business KPIs given the IT events.
4. **Application (APP):** This dataset is similar to the SER data, but it captures KPIs for the entire application instead of capturing at the service level. Even in this case, all business KPIs are treated as target variables and IT events are treated as exogenous variables and the goal of our forecasting is to predict the business KPIs given the IT events.

D TTM Model Hyper-parameters and Baselines

D.1 Pretraining: Pre-training is performed in a distributed fashion with 50 CPUs and 6 NVIDIA A100 GPUs. Standard model configurations are as follows: patch length pl = 64 (when sl is 512), 128 (when sl is 1024 or 1536) and 8 (when sl is 96); stride $s = pl$ (i.e., non-overlapping patches), number of patches $n = sl/pl$, number of levels in backbone L = 3, number of TTM blocks per level M = 2, number of decoder layers = 2, batch size b = 4500, number of epochs ep = 20, and dropout do = 0.4. PatchTSMixer-specific hyperparameters include feature scaler fs = 3, hidden feature size $hf = fs * pl$, expansion feature size $ef = hf * 2$. Please note that hf and n will change across TTM blocks based on the adaptive patching strategy. Resolution prefix tuning is enabled by default on all variants other than TTM_Q. Decoder channel-mixing and exogenous mixer blocks are disabled during pre-training and enabled during fine-tuning based on the dataset requirement.

D.2 Fine-tuning: Most model parameters remain the same from pretraining except the following parameters. Head dropout is changed during finetuning based on the target dataset used (0.7 for smaller ETT datasets and 0.2 for other datasets). Likewise, the batch size is set to 8 for Traffic, 32 for Electricity, and 64 for all other datasets. Moreover, decoder channel-mixing and exogenous mixer block are enabled for datasets that need cross-channel modelling (i.e. D2 datasets). Unlike pre-training, fine-tuning is executed in just 1 A100 GPU as it is a fast process. All these hyper-parameters are selected based on the validation performance, and the final test results are reported in the paper.

D.3 Computational Benefits of TTM over existing models - Setup details: Table 3 compares the computational benefits of TTM over existing TS-pretrained models and reports the following metrics: (i) GPU Inference Time per batch (in milliseconds (ms)), (ii) CPU Inference Time per batch (in seconds (s)), (iii) Max GPU Memory used during inference (in GB), (iv) Params: Total parameters of the models (in Millions). Experiments are conducted using sl = 512, fl = 96, and batch size = 32 in one A100 80GB GPU, 16 cores with 256GB memory. GPU is not enabled while capturing the CPU time. Since many pre-trained models process data in a purely univariate fashion, while TTM processes data in a multi-variate fashion, we set the number of channels c to 1 for this experiment so that, the number of samples per batch remains the same across all models for a fair comparison. In addition, we used a small batch size of 32 for this experiment, as many pre-trained models (like Chronos_L) were encountering out-of-memory (OOM) errors with high batch sizes. For this experiment, we set the number of probabilistic samples to 1 (i.e., `num_samples = 1`) for probabilistic algorithms (such as Chronos or Lag-Llama) to compute their fastest possible runtime. Note, that for forecast accuracy comparison, we set the number of samples to 100 for Lag-Llama and 20 for Chronos as suggested in their open-source code examples. All the baselines algorithms were evaluated using their open-sourced inference APIs as detailed in Section D.4. Please note that the computational benefits of TTM will further amplify if we use higher batch sizes or high number of channels as our models are extremely small and can process multiple channels at the same time using the channel-independence approach [22].

Category	Baseline	Used in Table	Results Generated From	Link to the used implementation
(a) TS pre-trained SOTA models	Moirai [35]	Zero-shot in Table 1 and 13	Table 6 and 22 of [35]	N/A
	Moirai [35]	Runtime in Table 3	Official implementation	`uni2ts`
	TimesFM [3]	Zero-shot in Table 1 and 13, Runtime in Table 3	Official implementation	`timesfm`
	Chronos [2]	Zero-shot in Table 2 and 15, Runtime in Table 3	Official implementation	`chronos-forecasting`
	Lag-Llama [26]	Zero-shot in Table 2 and 15, Runtime in Table 3	Official implementation	`lag-llama`
	Moment [10]	Full-shot Head-probing in Table 5 and 17	Table 2 of [10]	N/A
	Moment [10]	Runtime in Table 3	Official Implementation	`moment`
(b) LLM-based TS pre-trained models	GPT4TS [46]	Few-shot 5% in Table 4 and 16	Table 12 of [46]	N/A
	GPT4TS [46]	Runtime in Table 3, Exog. expt. in Table 6	Official Implementation	`One-Fits-All`
	GPT4TS [46]	Full-shot Head-probing in Table 5 and 17	Table 2 of [10]	N/A
	LLMTime [11]	Zero-shot in Table 20	Results available in the LLMTime Repository	`llmtime`
	Time-LLM [15]	Full-shot Head-probing in Table 5 and 17	Table 2 of [10]	N/A
	UniTime [18]	Zero-shot in Table 21	Table 5 of [18]	N/A
(b) Self-supervised pre-trained models	SimMTM [5] Ti-MAE [17] TST [42] LaST [34] TF-C [43] CoST [36] TS2Vec [40]	Table 22	Directly reported from SimMTM paper [5]	N/A
(d) Other SOTA Architectures	PatchTST [22]	Few-shot 5%in Table 4 and 16	Taken from [46]	N/A
	TSMixer [6]		Official Implementation	`PatchTSMixer`
	TimeMixer [33]		Official Implementation	`Time-Series-Library`
	iTransformer [19]		Official Implementation	`Time-Series-Library`
	TimesNet [37]		Taken from [46]	N/A
	Dinear [41]		Taken from [46]	N/A
	FEDFormer [45] Autoformer [38] Informer [44]	Full-shot end2end Table 17	Taken from [10]	N/A

Table 10: Implementation details for the baseline algorithms.

D.4 Baseline Implementation Details: We report the implementation details for all the baselines in Table 10.

E Sample Zero-shot Visualizations

Figure. 6 visualizes the zero-shot forecasts of TTM across different datasets illustrating the power of TTM to capture complex trends and seasonal patterns.

F Full Results Tables

Here, we present the complete versions of various tables in the main paper. These full versions essentially include the test results for multiple forecast lengths (fl) across all datasets. Occasionally, these results are averaged across forecast lengths to conserve space in the main paper.

F.1 Full table for all TTM variants: Table 11 and Table 12 captures the fine-grained results of all TTM variants (i.e. TTM_Q, TTM_B, $\text{TTM}_E$and TTM_A) on the $D1$ data benchmark set.

F.2 Full table for zero-shot experiment: Table 13 show the sliding window zero-shot results for all forecast lengths across all D1 datasets, and compares TTM variants with Moirai variants and TimesFM. Table 15 depicts the last-window zero-shot results for all forecast lengths across all of D1 datasets, and compares TTM with Chronos and Lag-Llama.

	FL	TTM_Q	TTM_B	TTM_E	TTM_A
ETTH1	96	0.365	0.364	0.363	**0.359**
	192	0.393	**0.386**	0.393	0.389
	336	0.415	**0.404**	0.406	0.405
	720	0.538	**0.424**	0.452	0.448
ETTH2	96	0.285	0.277	0.271	**0.264**
	192	0.341	0.334	0.324	**0.321**
	336	0.383	0.362	0.357	**0.351**
	720	0.441	0.408	**0.388**	0.395
ETTM1	96	0.413	0.322	0.327	**0.318**
	192	0.476	0.376	0.377	**0.354**
	336	0.553	0.407	0.395	**0.376**
	720	0.737	0.439	0.419	**0.398**
ETTM2	96	0.187	0.171	0.178	**0.169**
	192	0.261	0.238	0.238	**0.223**
	336	0.323	0.304	0.29	**0.276**
	720	0.436	0.41	0.379	**0.342**
Weather	96	**0.154**	0.158	0.166	0.159
	192	**0.203**	0.206	0.214	**0.203**
	336	0.256	0.256	0.254	**0.247**
	720	0.329	0.328	0.319	**0.314**
Electricity	96	0.169	0.166	0.157	**0.152**
	192	0.196	0.191	**0.174**	0.179
	336	0.209	0.207	0.195	**0.193**
	720	0.264	0.255	0.25	**0.243**
Traffic	96	0.518	0.514	0.476	**0.462**
	192	0.548	0.544	0.5	**0.491**
	336	0.55	0.575	0.51	**0.509**
	720	0.605	0.622	0.571	**0.547**
Model Size		**1M**	**1M**	**4M**	**5M**

Table 11: Zero-shot results of all TTM variants on D1 data benchmark across all sliding test windows (standard test protocol).

F.3 Other zero-shot comparisons: LLMTime [11] reported the test performance only on the last windows of the test datasets (instead of sliding windows) for horizons 96 and 192 due to computational reasons. We recreate the same experimental setup for TTM, and depict the comparative results in Table 20. We observe 26-36% improvement across all three variants of TTM with tremendous (70,000 to 14,000) reduction in model sizes. Another zero-shot comparison with the UniTime [18] model is shown in Table 21. In this comparison, TTM outperforms UniTime by 29-31%.

F.4 TTM Zero-shot *vs.* SOTA Full-shot (short context setting): In Table 14 we compare the zero-shot results of TTM variants with full-shot end-2-end training of popular TS architectures like iTransformer, PatchTST *etc.*. The full-shot SOTA algorithms were trained in short-context length setting ($sl = 96$) on the train split of each target dataset, and these results are obtained from the Moirai paper [35] where the authors draw similar comparison. TTM was tested in zero-shot setting without any training on the target datasets. We also provide the zero-shot results of Moirai variants and TimesFM from Table 1 for reference purpose. We can see that the zero-shot performance of all variants of TTM outperforms the full-shot performance of SOTA models, even though the latter are trained on the target datasets. This underscores the strength of the pre-trained TTM model.

F.5 Full table for 5% few-shot experiment: Table 16 shows the 5% few-shot results for all forecast lengths across all D1 datasets.

F.6 TTM *vs.* Cross-transfer models: Table 22 draws a comparative analysis of TTM_Q with SimMTM, Ti-MAE, TST, LaST, TF-C, CoST, and TS2Vec models in different few-shot settings (10% to 100% availability of training data) on ETTH1 dataset. The baseline models are trained on ETTH2 data, and tested on ETTH1 data, thus demonstrating their transferability across datasets having similar characteristics. The baseline numbers are taken from [5]. TTM_Q outperform all of them (including the recent SOTA SimMTM) by a significant margin. This highlights the usefulness of the pre-trained TTM weights and their ability to adapt to a target domain with few-shot fine-tuning.

	FL	TTM_Q	TTM_B	TTM_E	TTM_A
ETTH1	96	0.366	0.364	0.363	**0.359**
	192	0.391	**0.387**	0.393	0.394
	336	0.421	0.399	**0.398**	0.406
	720	-	-	-	-
ETTH2	96	0.282	0.277	0.271	**0.267**
	192	0.338	0.334	0.325	**0.321**
	336	0.383	0.361	0.357	**0.354**
	720	-	-	-	-
ETTM1	96	0.359	**0.313**	0.326	0.317
	192	0.402	0.357	0.371	**0.355**
	336	0.424	0.395	0.396	**0.374**
	720	0.575	0.437	0.418	**0.397**
ETTM2	96	0.174	0.171	0.178	**0.17**
	192	0.24	0.23	0.237	**0.222**
	336	0.299	0.293	0.284	**0.274**
	720	0.407	0.393	0.373	**0.345**
Weather	96	**0.152**	0.154	0.162	0.155
	192	**0.198**	0.203	0.215	0.201
	336	0.25	0.252	0.262	**0.244**
	720	0.326	0.327	0.319	**0.316**
Electricity	96	0.142	0.146	0.15	**0.141**
	192	0.162	0.164	0.171	**0.16**
	336	0.184	0.185	0.202	**0.179**
	720	**0.228**	0.236	0.303	0.24
Traffic	96	**0.401**	0.411	0.411	0.469
	192	0.425	**0.42**	0.44	0.488
	336	**0.437**	0.468	0.46	0.512
	720	-	-	-	-
Model Size		**1M**	**1M**	**4M**	**5M**

Table 12: Few-shot 5% results of all TTM variants on D1 data benchmark across all sliding test windows (standard test protocol).

F.7 Full table for full-shot head-probing experiment: Head Probing (HP) involves finetuning the pre-trained model heads on full data with backbone weights frozen. Table 17 compares the full-shot head-probing results of TTM with Moment, Time-LLM and GPT4TS. Table 18 and Table 19 compares the TTM Head probing results with the Full-shot End-To-End Training results of popular time-series architectures. It is important to note that end-to-end training updates the backbone weights, whereas head probing does not. TTM head probing results are either superior or highly competitive with other popular state-of-the-art methods that are trained end-to-end on the target data.

F.8 Full table: Impact of Adaptive Patching (AP): Table 23 shows the full table for studying the impact of adaptive patching across different amounts of pre-trained data settings. In both the settings, AP helps TTM to produce more accurate forecasts. However, the impact of AP is greater in the setting with a lesser amount of pre-training data, where there is more need to model at multiple granularities to compensate for the data size.

F.9 Full table: Impact of Resolution Prefix Tuning (RPT): Table 24 presents a comprehensive analysis of the impact of RPT on TTM. RPT generally enhances forecast performance, particularly when the pretraining (PT) data is abundant and diverse. In this scenario, incorporating a learnable resolution prefix token significantly benefits the models by allowing them to decouple the weights across resolutions effectively. Conversely, in setups with limited PT data where diversity challenges are minimal, RPT has a reduced impact. Additionally, we can see that RPT helps in scenarios when the context length is short. Table 25 shows the impact of RPT in shorter context length setting ($sl = 96$). We report the zero-shot results for $fl = 24$ in the table. In these scenarios, automatically detecting the resolution becomes a challenge for the model. Hence, by explicitly fusing the resolution information as a prefix, we can enhance the model's ability to learn effectively across resolutions.

	FL	TTM_B	TTM_E	TTM_A	$Moirai_S$	$Moirai_B$	$Moirai_L$	TimesFM
ETTH1	96	0.364	0.363	**0.359**	0.375	0.384	0.38	0.421
	192	**0.386**	0.393	0.389	0.399	0.425	0.44	0.472
	336	**0.404**	0.406	0.405	0.412	0.456	0.514	0.51
	720	0.424	0.452	0.448	**0.413**	0.47	0.705	0.514
ETTH2	96	0.277	0.271	**0.264**	0.281	0.277	0.287	0.326
	192	0.334	0.324	**0.321**	0.34	0.34	0.347	0.4
	336	0.362	0.357	**0.351**	0.362	0.371	0.377	0.434
	720	0.408	0.388	0.395	**0.38**	0.394	0.404	0.451
ETTM1	96	0.322	0.327	**0.318**	0.404	0.335	0.353	0.357
	192	0.376	0.377	**0.354**	0.435	0.366	0.376	0.411
	336	0.407	0.395	**0.376**	0.462	0.391	0.399	0.441
	720	0.439	0.419	**0.398**	0.49	0.434	0.432	0.507
ETTM2	96	0.171	0.178	**0.169**	0.205	0.195	0.189	0.205
	192	0.238	0.238	**0.223**	0.261	0.247	0.247	0.293
	336	0.304	0.29	**0.276**	0.319	0.291	0.295	0.366
	720	0.41	0.379	**0.342**	0.415	0.355	0.372	0.472
Weather	96	**0.158**	0.166	0.159	0.173	0.167	0.177	-
	192	0.206	0.214	**0.203**	0.216	0.209	0.219	-
	336	0.256	0.254	**0.247**	0.26	0.256	0.277	-
	720	0.328	0.319	**0.314**	0.32	0.321	0.365	-
Electricity	96	0.166	0.157	**0.152**	0.205	0.158	**0.152**	-
	192	0.191	0.174	0.179	0.22	0.174	**0.171**	-
	336	0.207	0.195	0.193	0.236	**0.191**	0.192	-
	720	0.255	0.25	0.243	0.27	**0.229**	0.236	-

Table 13: Zero-shot results of TTM over Moirai (ICML'24) and TimesFM (ICML'24). Electricity and Weather results for TimesFM are not reported as they were used by TimesFM for pretraining. Similarly, Traffic data was used in both Moirai and TimesFM pre-training, hence, skipped in this comparison.

	Zero-shot from Pre-Trained Models							**Full-shot End2End Training with short context length setting**							
Data	TTM_B	TTM_E	TTM_A	$Moirai_S$	$Moirai_B$	$Moirai_L$	TimesFM	iTransformer	TimesNet	PatchTST	Crossformer	TiDE	Dlinear	SCINet	FEDFormer
ETTH1	**0.394**	0.404	0.4	0.4	0.434	0.51	0.479	0.454	0.458	0.469	0.529	0.541	0.456	0.747	0.44
ETTH2	0.345	0.335	**0.333**	0.341	0.346	0.354	0.403	0.383	0.414	0.387	0.942	0.611	0.559	0.954	0.437
ETTM1	0.386	0.38	**0.362**	0.448	0.382	0.39	0.429	0.407	0.4	0.387	0.513	0.419	0.403	0.486	0.448
ETTM2	0.281	0.271	**0.252**	0.3	0.272	0.276	0.334	0.288	0.291	0.281	0.757	0.358	0.35	0.571	0.305
Weather	0.237	0.238	**0.231**	0.242	0.238	0.26	-	0.258	0.259	0.259	0.259	0.271	0.265	0.292	0.309
Electricity	0.205	0.194	0.192	0.233	0.188	0.188	-	**0.178**	0.193	0.216	0.244	0.252	0.212	0.268	0.214
TTM_B *f-imp(%)*				**6%** ↑	**1%** ↓	**4%** ↑	**15%** ↑	**4%** ↑	**7%** ↑	**7%** ↑	**34%** ↑	**22%** ↑	**15%** ↑	**37%** ↑	**13%** ↑
TTM_E *f-imp(%)*				**7%** ↑	**1%** ↑	**6%** ↑	**16%** ↑	**6%** ↑	**8%** ↑	**8%** ↑	**34%** ↑	**23%** ↑	**16%** ↑	**39%** ↑	**15%** ↑
TTM_A *f-imp(%)*				**10%** ↑	**4%** ↑	**9%** ↑	**19%** ↑	**9%** ↑	**11%** ↑	**11%** ↑	**36%** ↑	**26%** ↑	**19%** ↑	**40%** ↑	**17%** ↑

Table 14: Zero-shot Forecast-Improvement *(f-imp)* of TTM over Moirai (ICML'24), TimesFM (ICML'24) and other popular architectures (full shot trained with context length 96). MSE averaged across Fls: $\{96, 192, 336, 720\}$. Electricity and Weather results for TimesFM are not reported as they were used by TimesFM for pretraining. Similarly, Traffic data was used in both Moirai and TimesFM pre-training. Full-shot and Moirai results reported from [35], TimesFM results were generated from their released code.

G Model Insights and Explanation

G.1 Dataset preparation for TTM embedding analysis: To understand the representation obtained from TTM encoder, we have carried out a controlled analysis, using 3 datasets with varying observation frequency, viz., (1) weather (10 min), (2) electricity (1 hour), and (3) traffic (1 hour). We have selected three temporally distinct non-overlapping windows of length 1024 from each dataset (Figure 7). The selection criteria of these segments are distinct mean and standard deviation measures. From each of these segment, 512 context length windows are extracted in a rolling window fashion. Embedding vector from the encoder is collected at backbone output. Each of these representation tensors are flattened, and Principal Component Analysis (PCA) is carried out on the whole dataset. The project on the first two principle components is used to obtain the figure 5.

G.2 Channel Attention Map: The channel mixing block in the decoder of TTM consists of a gated attention block that produces an attention weight for each feature across channels. We have considered the mean attention weight across features and data samples to derive the feature contribution. We have used the model finetuned on the Bikesharing dataset for this purpose. Bikesharing data

	FL	TTM_B	TTM_E	TTM_A	$Chronos_T$	$Chronos_S$	$Chronos_B$	$Chronos_L$	Lag-llama
ETTH1	24	0.195	0.243	0.217	0.207	0.217	0.163	**0.161**	0.372
	48	**0.193**	0.244	0.226	0.33	0.277	0.228	0.226	0.376
	60	0.209	0.212	**0.2**	0.295	0.331	0.303	0.311	0.321
	96	**0.194**	0.209	0.207	0.424	0.383	0.272	0.298	0.299
	192	0.231	0.227	**0.221**	0.299	0.303	0.295	0.335	0.303
ETTH2	24	0.12	0.149	0.158	0.135	0.12	0.086	**0.065**	0.216
	48	0.151	0.18	0.197	0.186	0.207	0.194	**0.139**	0.187
	60	**0.166**	0.175	0.193	0.217	0.187	0.195	0.2	0.182
	96	**0.111**	0.13	0.155	0.197	0.156	0.22	0.207	0.135
	192	**0.105**	0.123	0.107	0.151	0.131	0.125	0.162	0.122
ETTM1	24	**0.141**	0.247	0.187	0.285	0.196	0.284	0.247	0.207
	48	**0.161**	0.241	0.182	0.704	0.343	0.207	0.311	0.881
	60	**0.153**	0.205	0.166	0.959	0.545	0.536	0.818	0.958
	96	**0.172**	0.209	0.184	0.854	0.589	0.508	0.538	1.006
	192	0.402	0.294	**0.232**	1.394	0.757	0.914	0.774	1.16
ETTM2	24	**0.04**	0.063	0.068	**0.04**	0.041	0.054	0.075	0.325
	48	**0.144**	0.16	0.152	0.379	0.253	0.236	0.198	0.337
	60	0.125	0.143	**0.119**	0.274	0.233	0.222	0.222	0.294
	96	0.137	0.108	**0.101**	0.147	0.12	0.171	0.192	0.314
	192	0.175	0.165	**0.146**	0.191	0.223	0.265	0.246	0.268
Weather	24	0.015	0.015	0.016	**0.008**	0.011	0.009	0.01	0.084
	48	0.018	0.017	0.026	0.024	0.017	**0.016**	0.022	0.105
	60	**0.024**	0.03	0.054	0.036	0.04	0.029	0.028	0.141
	96	0.034	**0.025**	0.053	0.042	0.068	0.05	0.049	0.057
	192	0.103	0.072	0.065	0.105	0.095	**0.047**	0.055	0.243
Electricity	24	0.409	0.421	0.433	0.463	0.429	0.42	0.409	**0.384**
	48	0.28	0.308	0.304	0.338	0.313	0.289	**0.278**	0.351
	60	**0.26**	0.287	0.279	0.346	0.301	0.279	0.273	0.363
	96	0.231	0.243	0.238	0.352	0.294	0.231	**0.227**	0.317
	192	0.495	0.495	**0.491**	0.616	0.548	0.503	0.507	0.551
Traffic	24	**0.231**	0.232	0.239	0.324	0.3	0.288	0.301	0.237
	48	0.193	**0.183**	0.187	0.244	0.232	0.213	0.226	0.192
	60	0.211	**0.191**	0.196	0.258	0.328	0.332	0.326	0.198
	96	0.21	0.199	0.215	0.256	0.219	0.217	**0.196**	0.213
	192	0.387	0.393	0.382	0.373	0.419	0.351	**0.297**	0.374

Table 15: Zero-shot results of TTM over Chronos (2024) and Lag-llama (2024) over the last test-window. Since Chronos and Lag-llama recommend/report results with shorter forecast lengths, we use different $FLs \in \{24, 48, 60, 96, 192\}$ in this experiment.

includes exogenous variables, viz. temperature, humidity, wind speed, etc. We analyzed the mean attention of the model across these exogenous variables for the forecast of rental bike count (Figure 5). As we observe, these attention weights highly correlate with the general data characteristics of his data, wherein - bike rentals are highly influenced by weather and holiday signals. Thus, TTM fine-tuning process is quick as well as explainable.

H Limitations and Future Work

TTM is currently focused solely on forecasting tasks, similar to other forecast pretraining models like Moirai [35], Chronos [2], and TimesFM [3]. However, recent models such as Moment and GPT4TS are taking initial steps to expand their capabilities across multiple downstream tasks, including classification, regression, and anomaly detection. Inspired by these advancements, we plan to extend TTM's functionality to encompass a broader range of downstream tasks.

Another limitation of TTM is the need to train different models for different context length settings. Due to its non-transformer-based architecture, TTM is sensitive to context lengths. Consequently, in this paper, we introduce three variants of TTM, each optimized for different context length settings. Looking ahead, we aim to enhance TTM's backbone to automatically adapt to dynamically varying context lengths.

In addition, existing pre-trained models like lag-llama, Moirai support probabilistic forecasting while TTM currently supports only point forecasting. We plan to extend TTM with distribution heads to facilitate probabilistic forecasts in future work.

Data		Pre-trained Models					Other popular architectures					
	FL	TTM_B	TTM_E	TTM_A	GPT4TS	Time-LLM	PatchTST	TSMixer	TimeMixer	iTransformer	TimesNet	Dlinear
ETTH1	96	0.364	0.363	**0.359**	0.543	0.483	0.557	0.554	0.899	0.674	0.892	0.547
	192	**0.387**	0.393	0.394	0.748	0.629	0.711	0.673	0.942	0.757	0.94	0.72
	336	0.399	**0.398**	0.406	0.754	0.768	0.816	0.678	1.423	0.838	0.945	0.984
	720	-	-	-	-	-	-	-	-	-	-	-
ETTH2	96	0.277	0.271	**0.267**	0.376	0.336	0.401	0.348	0.356	0.382	0.409	0.442
	192	0.334	0.325	**0.321**	0.418	0.406	0.452	0.419	0.549	0.445	0.483	0.617
	336	0.361	0.357	**0.354**	0.408	0.405	0.464	0.389	0.619	0.483	0.499	1.424
	720	-	-	-	-	-	-	-	-	-	-	-
ETTM1	96	**0.313**	0.326	0.317	0.386	0.316	0.399	0.361	0.515	0.437	0.606	0.332
	192	0.357	0.371	**0.355**	0.44	0.45	0.441	0.411	0.535	0.49	0.681	0.358
	336	0.395	0.396	**0.374**	0.485	0.45	0.499	0.467	0.621	0.563	0.786	0.402
	720	0.437	0.418	**0.397**	0.577	0.483	0.767	0.677	0.64	0.78	0.796	0.511
ETTM2	96	0.171	0.178	**0.17**	0.199	0.174	0.206	0.2	0.231	0.217	0.22	0.236
	192	0.23	0.237	0.222	0.256	**0.215**	0.264	0.265	0.313	0.266	0.311	0.306
	336	0.293	0.284	0.274	0.318	**0.273**	0.334	0.314	0.356	0.322	0.338	0.38
	720	0.393	0.373	**0.345**	0.46	0.433	0.454	0.41	0.46	0.43	0.509	0.674
Weather	96	**0.154**	0.162	0.155	0.175	0.172	0.171	0.188	0.187	0.211	0.207	0.184
	192	0.203	0.215	**0.201**	0.227	0.224	0.23	0.234	0.324	0.269	0.272	0.228
	336	0.252	0.262	**0.244**	0.286	0.282	0.294	0.287	0.307	0.316	0.313	0.279
	720	0.327	0.319	**0.316**	0.366	0.366	0.384	0.365	0.451	0.393	0.4	0.364
Electricity	96	0.146	0.15	**0.141**	0.143	0.147	0.145	0.147	0.155	0.157	0.315	0.15
	192	0.164	0.171	0.16	0.159	**0.158**	0.163	0.172	0.219	0.181	0.318	0.163
	336	0.185	0.202	0.179	0.179	0.178	**0.175**	0.19	0.273	0.219	0.34	**0.175**
	720	0.236	0.303	0.24	0.233	0.224	**0.219**	0.28	0.309	0.249	0.635	**0.219**
Traffic	96	0.411	0.411	0.469	0.419	0.414	**0.404**	0.408	0.463	0.43	0.854	0.427
	192	0.42	0.44	0.488	0.434	0.419	**0.412**	0.421	0.548	0.445	0.894	0.447
	336	0.468	0.46	0.512	0.449	**0.437**	0.439	0.477	0.498	0.481	0.853	0.478
	720	-	-	-	-	-	-	-	-	-	-	-

Table 16: TTM Few-shot 5% MSE reported across all the standard FLs considered. TTM, Pre-trained baselines and other model architectures are trained with 5% train data.

Data		Head Probing of Pre-Trained Models						Full-shot end-2-end Training with 512 context length						
	FL	TTM_B	TTM_E	TTM_A	Moment	GPT4TS	Time-LLM	TimeMixer	PatchTST	TimesNet	TSMixer	FEDFormer	Autoformer	Informer
ETTH1	96	**0.36**	0.362	0.363	0.387	0.376	0.408	0.361	0.37	0.384	0.368	0.376	0.449	0.865
	720	**0.436**	0.449	0.442	0.454	0.477	0.523	0.445	0.447	0.521	0.444	0.506	0.514	1.181
ETTH2	96	0.269	0.273	**0.262**	0.288	0.285	0.285	0.271	0.274	0.34	0.276	0.346	0.358	3.755
	720	0.39	0.402	0.392	0.403	0.406	0.399	**0.342**	0.379	0.462	0.395	0.463	0.515	3.647
ETTM1	96	0.291	0.293	**0.283**	0.293	0.292	0.384	0.291	0.29	0.338	0.291	0.379	0.505	0.672
	720	0.419	0.408	**0.393**	0.405	0.417	0.437	0.415	0.416	0.478	0.416	0.543	0.671	1.166
ETTM2	96	0.164	**0.158**	**0.158**	0.17	0.173	0.181	0.164	0.165	0.187	0.164	0.203	0.255	0.365
	720	0.35	**0.347**	0.369	0.363	0.378	0.366	0.359	0.362	0.408	0.358	0.421	0.433	3.379
Weather	96	**0.146**	0.154	0.149	0.154	0.162	-	0.147	0.149	0.172	**0.146**	0.217	0.266	0.3
	720	0.323	0.324	0.318	0.315	0.326	-	**0.31**	0.314	0.365	0.316	0.403	0.419	1.059
Electricity	96	0.129	0.129	**0.128**	0.138	0.139	-	0.129	0.129	0.168	0.129	0.193	0.201	0.274
	720	0.2	0.193	0.191	0.211	0.206	-	0.194	0.197	0.22	**0.186**	0.246	0.254	0.373
Traffic	96	0.368	0.372	**0.352**	0.391	0.388	-	0.36	0.36	0.593	0.356	0.587	0.613	0.719
	720	**0.431**	0.425	0.419	0.45	0.45	-	0.43	0.432	0.64	0.424	0.626	0.66	0.864

Table 17: Head Probing (HP) involves finetuning the pre-trained model heads on full data with backbone weights frozen. Head probing results of TTM are compared with the Head probing results of other Pre-trained models and also with the Full-shot end-to-end training of popular TS architectures. TTM's Head probing results consistently outperform other HP benchmarks and also very competitive as compared to the full end-to-end training of popular TS architectures. MSE across FLs (96,720) are reported from [10]. Time-LLM results for large datasets are not reported in [10] due to computational issues. It is important to note that end-to-end training updates the backbone weights, whereas head probing does not.

Data		**TTM Head Probing**				**Full-shot end-2-end Training with 512 context length**								
	FL	TTM_Q	TTM_B	TTM_E	TTM_A	TimeMixer	TSMixer	PatchTST	TimesNet	CrossFormer	Dlinear	FEDFormer	Autoformer	Informer
ETTH1	96	0.373	**0.36**	0.362	0.363	0.361	0.368	0.37	0.384	0.418	0.375	0.376	0.449	0.865
	720	**0.424**	0.436	0.449	0.442	0.445	0.444	0.447	0.521	0.733	0.472	0.506	0.514	1.181
	192	0.398	**0.392**	0.394	**0.392**	0.409	0.399	0.413	0.436	0.539	0.405	0.42	0.5	1.008
	336	**0.397**	0.401	0.403	0.413	0.43	0.421	0.422	0.638	0.709	0.439	0.459	0.521	1.107
ETTH2	96	0.283	0.269	0.273	**0.262**	0.271	0.276	0.274	0.34	0.425	0.289	0.346	0.358	3.755
	720	0.417	0.39	0.402	0.392	**0.342**	0.395	0.379	0.462	0.775	0.605	0.463	0.515	3.647
	192	0.328	0.336	0.325	0.324	0.317	0.33	0.314	**0.231**	0.473	0.383	0.429	0.456	5.602
	336	0.361	0.359	0.356	0.351	0.332	0.357	**0.329**	0.452	0.581	0.448	0.496	0.482	4.721
ETTM1	96	0.286	0.291	0.293	**0.283**	0.291	0.291	0.293	0.338	0.361	0.299	0.379	0.505	0.672
	720	0.417	0.419	0.408	**0.393**	0.415	0.416	0.416	0.478	0.703	0.425	0.543	0.671	1.166
	192	0.333	**0.325**	0.335	0.332	0.327	0.333	0.333	0.374	0.387	0.335	0.426	0.553	0.795
	336	0.364	0.363	0.364	**0.353**	0.36	0.365	0.369	0.41	0.605	0.369	0.445	0.621	1.212
ETTM2	96	0.165	0.164	**0.158**	**0.158**	0.164	0.164	0.166	0.187	0.275	0.167	0.203	0.255	0.365
	720	0.358	0.35	**0.347**	0.369	0.359	0.358	0.362	0.408	1.208	0.397	0.421	0.433	3.379
	192	0.22	0.219	0.215	**0.213**	0.223	0.219	0.223	0.249	0.345	0.224	0.269	0.281	0.533
	336	0.269	0.277	**0.26**	0.269	0.279	0.273	0.274	0.321	0.657	0.281	0.325	0.339	1.363
Weather	96	**0.144**	0.146	0.154	0.149	0.147	0.146	0.149	0.172	0.232	0.176	0.217	0.266	0.3
	720	0.317	0.323	0.324	0.318	**0.31**	0.316	0.314	0.365	0.526	0.323	0.403	0.419	1.059
	192	0.19	0.19	0.207	0.192	**0.189**	0.191	0.194	0.219	0.371	0.22	0.276	0.307	0.598
	336	0.247	0.242	0.25	**0.24**	0.241	0.243	0.306	0.246	0.495	0.265	0.339	0.359	0.578
Electricity	96	0.13	0.129	0.129	**0.128**	0.129	0.129	0.129	0.168	0.15	0.14	0.193	0.201	0.274
	720	0.202	0.2	0.193	0.191	0.194	**0.186**	0.197	0.22	0.251	0.203	0.246	0.254	0.373
	192	0.149	0.149	0.148	0.144	**0.14**	0.146	0.147	0.184	0.161	0.153	0.201	0.222	0.296
	336	0.164	0.163	0.161	0.162	0.161	**0.158**	0.163	0.198	0.182	0.169	0.214	0.213	0.3
Traffic	96	0.367	0.368	0.372	**0.352**	0.36	0.356	0.36	0.593	0.514	0.41	0.587	0.613	0.719
	720	0.432	0.431	0.425	**0.419**	0.43	0.424	0.432	0.64	0.573	0.466	0.626	0.66	0.864
	192	0.387	0.403	0.365	**0.359**	0.375	0.377	0.379	0.617	0.549	0.423	0.604	0.616	0.696
	336	0.414	0.395	0.379	**0.375**	0.385	0.385	0.392	0.629	0.53	0.436	0.621	0.622	0.777

Table 18: **Full view: TTM Head probing Vs Full-shot End-To-End Training** of popular time-series architectures reported for all FLs. Head Probing (HP) involves finetuning the pre-trained model heads on full data with backbone weights frozen. TTM head probing results are either superior or highly competitive with other popular state-of-the-art methods that are trained end-to-end on the target data. It is important to note that end-to-end training updates the backbone weights, whereas head probing does not.

Data	**TTM Head Probing**				**Full-shot end-2-end Training with 512 context length**								
Data	TTM_Q	TTM_B	TTM_E	TTM_A	TimeMixer	TSMixer	PatchTST	TimesNet	CrossFormer	Dlinear	FEDFormer	Autoformer	Informer
ETTH1	0.398	**0.397**	0.402	0.402	0.411	0.408	0.413	0.495	0.6	0.423	0.44	0.496	1.04
ETTH2	0.347	0.338	0.339	0.332	**0.316**	0.34	0.324	0.371	0.564	0.431	0.434	0.453	4.431
ETTM1	0.35	0.35	0.35	**0.34**	0.348	0.351	0.353	0.4	0.514	0.357	0.448	0.588	0.961
ETTM2	0.253	0.252	**0.245**	0.252	0.256	0.254	0.256	0.291	0.621	0.267	0.304	0.327	1.41
Weather	0.224	0.225	0.234	0.225	**0.222**	0.224	0.241	0.25	0.406	0.246	0.309	0.338	0.634
Electricity	0.161	0.16	0.158	0.156	0.156	**0.155**	0.159	0.192	0.186	0.166	0.214	0.222	0.311
Traffic	0.4	0.399	0.385	**0.376**	0.388	0.386	0.391	0.62	0.542	0.434	0.609	0.628	0.764

Table 19: **Average view: TTM Head probing Vs Full-shot End-To-End Training** of popular time-series architectures averaged across FLs 96,192,336,720. Head Probing (HP) involves finetuning the pre-trained model heads on full data with backbone weights frozen. TTM head probing results are either superior or highly competitive with other popular state-of-the-art methods that are trained end-to-end on the target data. It is important to note that end-to-end training updates the backbone weights, whereas head probing does not.

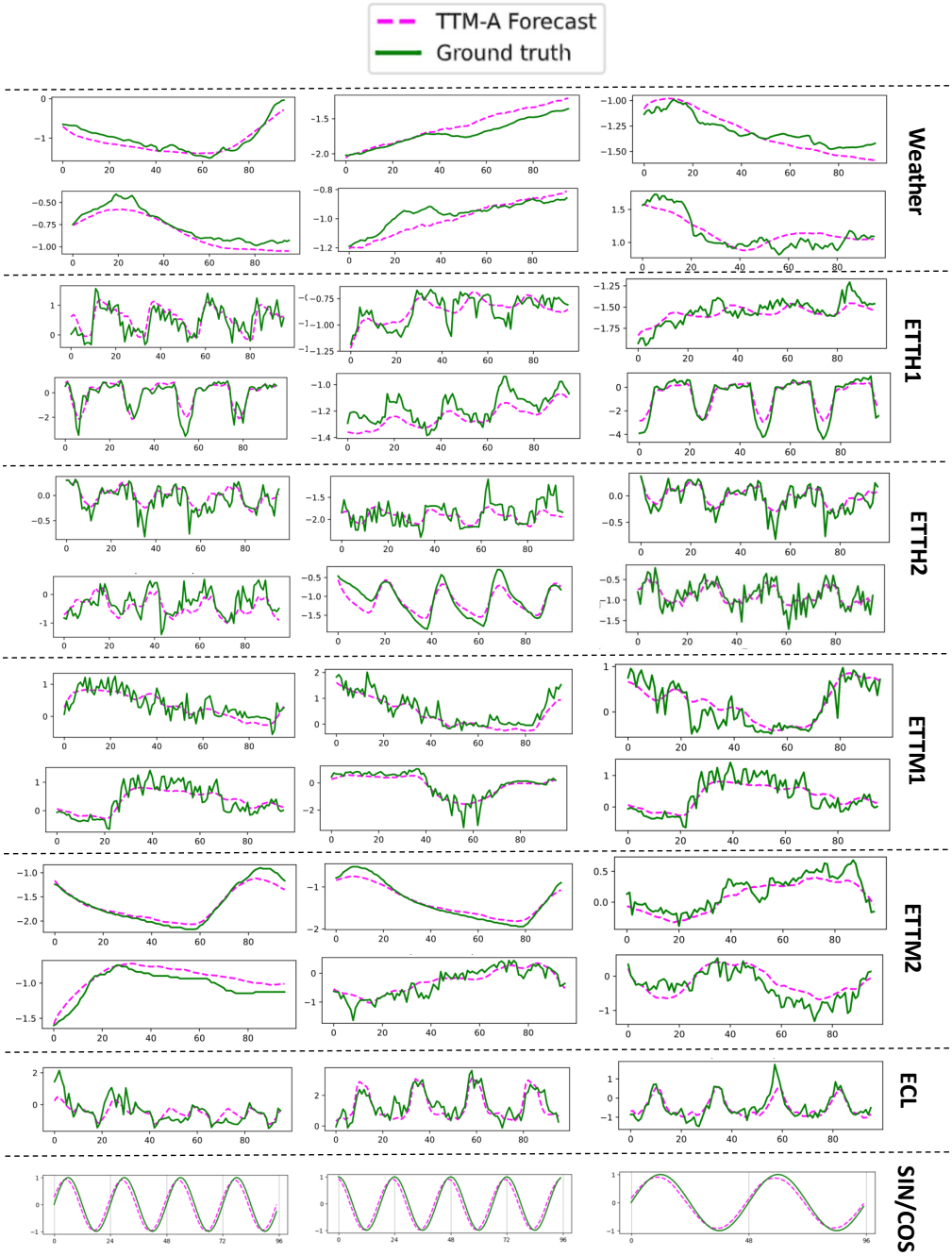

Figure 6: Sample TTM Zero-shot Forecasts across datasets

	FL	TTM_B	TTM_E	TTM_A	LLMTime
ETTM2	96	0.137	0.108	**0.101**	0.167
	192	0.175	0.165	**0.146**	0.198
Weather	96	0.034	**0.025**	0.053	0.107
	192	0.103	0.072	0.065	**0.062**
Electricity	96	**0.231**	0.243	0.238	0.609
	192	0.495	0.495	**0.491**	0.96
Traffic	96	0.21	**0.199**	0.215	0.34
	192	0.387	0.393	**0.382**	0.526
Model Size		**1M**	**4M**	**5M**	**70B**
TTM_B *f-imp(%) s-imp(X)*					**26%** ↑ **70000X** ↑
TTM_E *f-imp(%) s-imp(X)*					**36%** ↑ **17500X** ↑
TTM_A *f-imp(%) s-imp(X)*					**36%** ↑ **14000X** ↑

Table 20: LLM-Time Vs TTM: Zeroshot MSE reported on last test window set. Results reported in LLMTime [11] are used for this comparison.

	FL	TTM_B	TTM_E	TTM_A	Unitime
ETTH2	96	0.277	0.271	**0.264**	0.306
	192	0.334	0.324	**0.321**	0.389
	336	0.362	0.357	**0.351**	0.424
	720	0.408	**0.388**	0.395	0.434
Weather	96	**0.158**	0.166	0.159	0.21
	192	0.206	0.214	**0.203**	0.264
	336	0.256	0.254	**0.247**	0.326
	720	0.328	0.319	**0.314**	0.402
Electricity	96	0.166	0.157	**0.152**	0.409
	192	0.191	**0.174**	0.179	0.41
	336	0.207	0.195	**0.193**	0.439
	720	0.255	0.25	**0.243**	0.487
TTM_B f-imp(%)		29%			
TTM_E f-imp(%)		30%			
TTM_A f-imp(%)		31%			

Table 21: TTM vs UniTime MSE Improvement (*f-imp*) in zero-shot setting using full sliding-window test set. Results reported in UniTime [18] are used for this comparison.

	10%	25%	50%	75%	100%	**IMP**
TTM_Q	**0.422**	**0.421**	**0.413**	**0.402**	**0.398**	-
SimMTM	0.591	0.535	0.491	0.466	0.415	**17%**
Ti-MAE	0.660	0.594	0.55	0.475	0.466	**24%**
TST	0.783	0.641	0.525	0.516	0.469	**28%**
LaST	0.645	0.610	0.540	0.479	0.443	**23%**
TF-C	0.799	0.736	0.731	0.697	0.635	**43%**
CoST	0.784	0.624	0.540	0.494	0.428	**26%**
TS2Vec	0.655	0.632	0.599	0.577	0.517	**31%**

Table 22: Cross transfer learning MSE improvement (IMP) for self-supervised pre-training methods in various few-shot settings (10%,25%,50%,75%,100%).

	Less PT Data (250M samples)		More PT Data (1Bsamples)	
Data	w/o AP	w/ AP	w/o AP	w/ AP
ETTH1	0.369	**0.365**	0.367	**0.364**
ETTH2	**0.283**	0.285	**0.275**	0.277
ETTM1	0.446	**0.413**	0.326	**0.322**
ETTM2	0.191	**0.187**	0.172	**0.171**
Weather	0.159	**0.154**	0.161	**0.158**
Electricity	0.179	**0.169**	0.174	**0.166**
Traffic	0.521	**0.518**	0.522	**0.514**
f-imp(%)	**3%**		**1.5%**	

Table 23: Impact of Adaptive Patching(AP) in less pre-training (PT) and more pre-training (PT) data setting. Zero-shot results on FL 96 reported. AP generally improves the forecasting accuracy across both setups, but the impact is more when PT data is less as AP enables modelling at different resolutions in different layers of the model. ['w/': with, 'w/o': 'without'.]

	Less PT Data (250M samples)		More PT Data (1B samples)	
Data	w/o RPT	w/ RPT	w/o RPT	w/ RPT
ETTH1	**0.365**	0.36	0.366	**0.364**
ETTH2	0.285	**0.28**	0.285	**0.277**
ETTM1	0.413	**0.384**	0.341	**0.322**
ETTM2	**0.187**	0.194	0.18	**0.171**
Weather	**0.154**	0.16	**0.153**	0.158
Electricity	**0.169**	0.175	0.178	**0.166**
Traffic	**0.518**	**0.518**	0.528	**0.514**
f-imp(%)	**0%**		**3%**	

Table 24: Impact of Resolution Prefix Tuning (RPT) in less pre-training (PT) and more pre-training (PT) data setting. Zero-shot results on FL 96 reported. RPT generally enhances the forecast performance especially when the volume and diversity in the pretraining (PT) data are high. In this setting, adding a learnable resolution prefix token greatly helps, as it enables the models to easily decouple the weights across resolutions. However, in less PT setup where the challenges in diversity modelling are not observed, RPT does not have much impact. ['w/': with, 'w/o': 'without'.]

	$\textbf{TTM}_{Q}$ SL **= 96:** FL **= 24**	
	w/o RPT	**w/ RPT**
ETTH1	0.373	**0.358**
ETTH2	0.180	**0.179**
ETTM1	0.559	**0.387**
ETTM2	0.127	**0.108**
Weather	**0.103**	**0.103**
Electricity	0.208	**0.201**
Traffic	0.754	**0.740**
IMP (%)	**8%**	

Table 25: Impact of RPT in less context setting (SL = 96). Zero-shot results on FL 24 reported. RPT helps in scenarios when the context length (sl) is short. In these scenarios, automatically detecting the resolution becomes a challenge for the model. Hence, by explicitly fusing the resolution information as a prefix, we can enhance the model's ability to learn effectively across resolutions. ['w/': with, 'w/o': 'without'.]

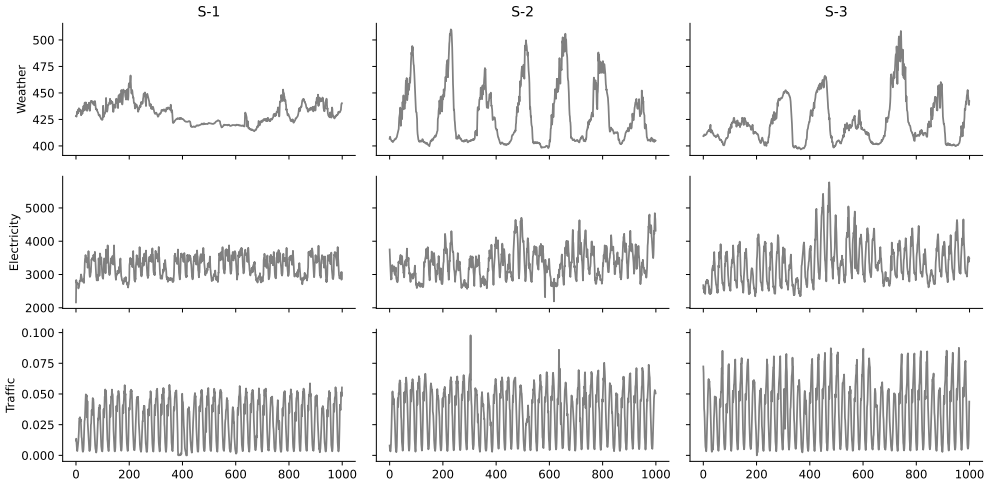

Figure 7: Data segments for TTM embedding analysis.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes. The main claims made in the abstract and introduction (Section 3) accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix Section H explains the limitations of this work and future directions.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Not applicable, as this work is grounded more on large-scale experimentation and empirical analysis.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide detailed information on the full experimental setup, model hyperparameters and dataset details. We also provide information on how each result of every baseline is reported. Refer to Section D, C.2, C.1, D.4 for more details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Yes. The source code and model weight links for reproducibility are shared in the abstract.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The relevant details are provided in Section D

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The proposed approach and many of the associated baseline papers do not report error bars as our experiments fall under Foundation Models, which are computationally very expensive to pre-train for multiple seeds. However, TTM is compared with other state-of-the-art models across multiple settings (4 different forecast lengths and 3 different variants), wherein, TTM outperforms the baselines consistently in all these experiments, to give substantial evidence for our claims.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experimental section has all the relevant details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the details and we conform to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Not Applicable, as our work does not directly relate to the societal impacts in the ecosystem.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Not applicable to our work, as no such misuse has been reported.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, we followed the licensing and terms of use very carefully while architecting our design. All assets used in this work will be credited properly.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes, all source code files and associated scripts are well documented and will further be improved before open-source release.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Not applicable to our work because no human participants were involved.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Not applicable to our work because no human participants were involved.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.