PTEF: EFFICIENT LONG-TERM TIME SERIES FORECASTING VIA LAYER PRUNED TEXT EMBEDDINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Time series forecasting plays a crucial role across diverse industries. Existing LLM-based approaches suffer from large model sizes and significant computational costs that limit their applicability in resource-constrained environments. We propose Pruned Text Embeddings for Time Series Forecasting (PTEF), a lightweight architecture using frozen pruned text embedding models to encode numerical sequences, exploiting implicit numerical and structural patterns acquired from large text corpora, as in zero-shot text-to-number forecasting. PTEF applies channel-independence to treat each input channel as a standalone sequence, encodes it via a frozen pruned embedding model in a single forward pass, and passes the resulting embedding vectors through a compact CNN-based forecasting head. This design decouples representation learning from forecasting, reducing trainable parameters to just 0.094M. In few-shot settings using only 10% of training data, PTEF achieves the best average ranking across long-term forecasting benchmarks, outperforming larger LLM-based models including Time-LLM and LLM4TS, while remaining competitive in full-shot settings despite its significantly smaller capacity. Using only a small embedding models such as all-MiniLM-L6-v2, PTEF demonstrates strong generalisation across multiple embedding architectures. Ablation studies confirm that pre-trained text embeddings are the primary driver of PTEF's performance, validating the effectiveness of cross-modal transfer from text to numerical time series.

1 INTRODUCTION

Time Series Forecasting (TSF) plays a critical role across a wide range of domains, from traffic management and weather prediction to energy consumption and healthcare monitoring. In many of these settings, labelled training data is scarce. This scarcity constrains purely supervised approaches and motivates the search for methods that can generalise from small datasets. In the recent years Large Language Models (LLMs) have expanded into time series analysis and forecasting. Researchers used rich representations learned by LLMs and transferred them to temporal prediction tasks. Some examples include: LLM4TS (Chang et al., 2025) - employs two stage adaptation, Time-LLM (Jin et al., 2024) - uses prompting as prefix and TEMPO (Cao et al., 2024) - leverages time series decomposition with dynamic prompt adaptation. While these models achieve competitive performance on standard benchmarks, they come with significant drawbacks. Their large model sizes, high parameter counts and heavy computational demands for both training and inference limit applicability in low-resource environments. Despite their scale, performance gains in low-data settings remain inconsistent (Chang et al., 2025; Jin et al., 2024), raising a key question: *Can useful NLP representations be transferred to TSF without the overhead of a full LLM backbone?*

We answer this by proposing **PTEF** (**P**runed **T**ext **E**mbedding for time series **F**orecasting), a lightweight architecture that leverages frozen layer pruned pre-trained text embedding models to encode formatted numerical sequences. The core intuition, grounded in prior findings demonstrating that text-trained models can natively process numerical data (Gruver et al., 2023), demonstrated that formatting time series values as structured text allows pre-trained embedding models to transfer implicit numerical patterns, periodicity, and temporal structure to forecasting tasks, without any domain-specific fine-tuning. PTEF extends this cross-modal transfer paradigm beyond the classification Kaur et al. (2025) to the more challenging and important long-term forecasting.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

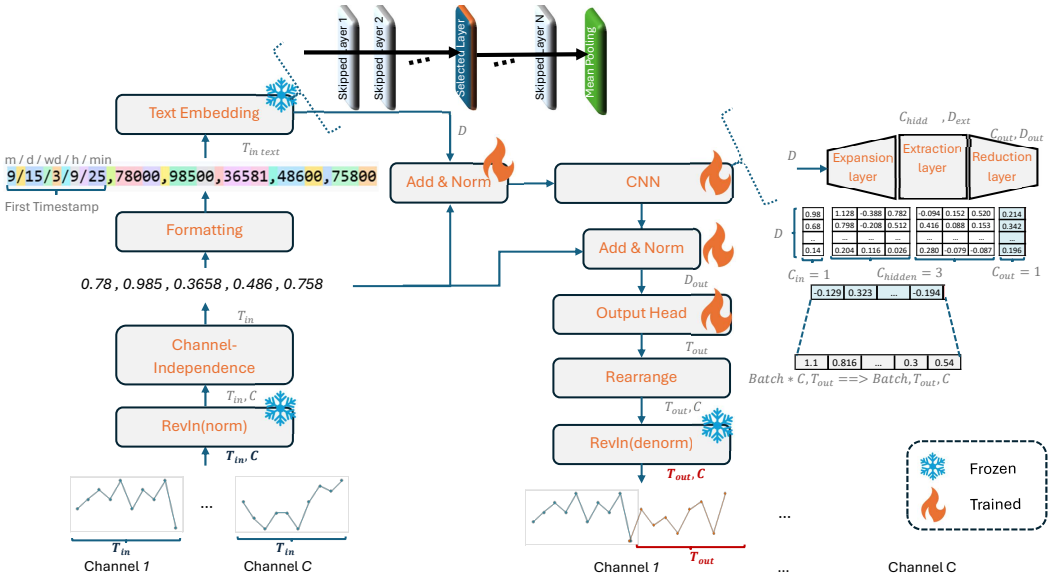

Figure 1: **PTEF Architecture** first normalizes the time series data using reversible instance normalization, concatenates it with the first timestamp (left side of the figure), formats it into a text representation and generates embeddings for it. The embedding vectors are then combined with the original time series using point-wise addition. A small forecasting model, consisting of a CNN followed by an linear projection (right side of the figure), is used to perform the final horizon prediction. During training, only the CNN and linear projection components are trained. An example is provided to illustrate the time series data transformation throughout the architecture.

PTEF is composed of two core stages. First, each input channel is treated independently, formatted as structured text and encoded by a pruned frozen embedding model in a single forward pass. Second, the resulting embeddings are passed through a compact CNN-based forecasting head to produce horizon predictions. This design separates representation learning from forecasting, keeping the number of trainable parameters to just 0.094M while preserving strong predictive performance. Figure 1 illustrates the overall architecture and the parameter efficiency advantage of PTEF relative to existing LLM-based baselines. Our main contributions are:

– We propose **PTEF** architecture that efficiently adapt standalone frozen layer pruned pre-trained text embedding models, without relying on full LLM backbones, for long-term TSF.

– We demonstrate that, **in few-shot settings using only 10% of training data**, PTEF achieves the top average ranking across five standard long-term forecasting benchmarks, outperforming larger LLM-based models including Time-LLM and LLM4TS, while using only 0.094M trainable parameters, just 0.68% of total model parameters.

– Through ablation studies across multiple embedding architectures of different sizes, we confirm that pruned pre-trained text embeddings are the primary driver of PTEF's performance, demonstrating the robustness and generalisability of the approach across diverse forecasting benchmarks.

2 Related Work

2.1 Transfer Learning from NLP to TSF

Recent TSF advances have adapted pre-trained LLMs by transferring parameter weights and cross-domain reasoning to temporal data (Kottapalli et al., 2025). This takes three forms: frozen backbones - One Fits All (Zhou et al., 2023) retains GPT-2 weights, Time-LLM (Jin et al., 2024) uses prompt-as-prefix reprogramming, TEMPO (Cao et al., 2024) applies dynamic prompt adaptation, and TEST (Sun et al., 2024) maps temporal data into the LLM's semantic space; multi-stage fine-tuning, as in LLM4TS (Chang et al., 2025); and zero-shot text-to-number prediction, demonstrated by LLMTime (Gruver et al., 2023). While fine-tuning gives a better domain adaptation, its computational cost motivates frozen backbone approaches, the setting PTEF is designed to excel in.

2.2 LONG-TERM TSF (LTSF)

LTSF has evolved from statistical baselines like ARIMA (Box & Jenkins, 1970) and VAR (Sims, 1980) to Transformer-based models including Informer (Zhou et al., 2021), Autoformer (Wu et al., 2021), and FEDformer (Zhou et al., 2022). TimesNet (Wu et al., 2023b) transforms 1D time series into 2D tensors and process them using vision-based architectures. Later, DLinear (Zeng et al., 2023) challenged these complex models. It showed that simple linear layers could actually outperform early Transformers, setting a strong new baseline. To improve Transformers, PatchTST (Nie et al., 2023) introduced two key ideas: dividing the data into small patches and using a Channel-Independent (CI) approach. From Similarity to Superiority (Chen et al., 2024) introduced the Channel Clustering Module (CCM), it groups dynamically channels based on similarities to balance the trade-off between channel-independent and channel-dependent modeling. The Capacity and Robustness Trade-Off (Han et al., 2024) demonstrates that channel-independent forecasting models outperform channel-dependent because they are more robust to distribution shifts. iTransformer (Liu et al., 2024) arguing that the relationships between different variables are too important to ignore, they proposed a new approach. Instead of applying attention over time, it treats the entire history of a single variable as one token and applies attention across the different channels. This proved that mixing channels can still achieve excellent results if the model is structured correctly.

2.3 TEXT EMBEDDING FOR TSF

Text embeddings serve as a fundamental mechanism for compressing high-dimensional information into dense semantic vectors, evolving from dual-encoder architectures like Sentence-BERT (Reimers & Gurevych, 2019) to advanced prefix-based frameworks like E5 (Wang et al., 2022). These models effectively maintains numerical relationships by mapping numbers to a vector space where geometric positions and distances consistently reflect their relative magnitudes. Similarly, time series data contains trend and seasonality, but training robust foundational embeddings from scratch remains challenging due to data scarcity. Recent research has explored projecting temporal data into established, pre-trained text spaces for classification (Kaur et al., 2025), which formats numerical data into strings and fuses embeddings from a frozen off-the-shelf text model with raw temporal data. TEST (Sun et al., 2024) successfully aligns temporal features with discrete text prototypes. However, using large text embedding models for forecasting introduces high inference latency and computational cost. Authors in the study (Nematirad et al., 2025) demonstrated that stripping embedding layers often improves efficiency, but the true potential of data embeddings when utilized as a main architectural component remains unassessed. To address this, we introduce PTEF, a long-term forecasting framework that leverages lightweight pruned text embedding architectures. By prioritizing small-scale models, PTEF dramatically reduces computational overhead while achieving state-of-the-art performance, particularly in few-shot scenarios.

2.4 LAYER PRUNING

Layer pruning is a common approach to reduce computational cost of models by removing less important layers. (Fan et al., 2020; Sajjad et al., 2020; He et al., 2024) showed that transformer layers can be removed during inference, and often maintain strong performance. (Men et al., 2025) showed that many transformer layers contain redundant information, (Zhao et al., 2025) demonstrated that different tokens require varying amounts of context and processing depth, making layer pruning an effective way to reduce computation while preserving model performance.

3 METHODOLOGY

As shown in **Figure 1**, the PTEF consists of four components: i) Preprocessing, ii) Sequence Embedding, iii) Raw Sequences Residual and iv) Forecasting Head.

3.1 PREPROCESSING

In Multivariate TSF, model takes an input of shape (T_{in}, C) and predicts an output of shape (T_{out}, C), where T_{in} is the look-back window, T_{out} the forecasting horizon, and C the number of channels.

Reversible Instance Normalization (RevIn): The RevIn (Kim et al., 2022) allows the models to focus on learning the trends, seasonality, and the other variation behavior that characterize the data without learning the amplitude. The concept proved to be effective in improving performance. In our case, we calculate μ and σ for each channel independently and add them later to the predicted values.

$$Z_{norm} = \frac{Z_{input} - \mu}{\sqrt{\sigma + \epsilon}}, \hat{Y} = \hat{Y}_{predicted} \cdot \sqrt{\sigma + \epsilon} + \mu$$

Where Z_{input} is the input and $\hat{Y}$ is the output.

Channel Independence (CI): Input sample $Z_{norm} \in \mathbb{R}^{T_{in} \times C}$ contains a look-back window of size T_{in} and C channels, we applied CI from PatchTST (Nie et al., 2023). The channel dimension is merged into the batch dimension, forcing the model to treat each channel as a standalone sample and capturing inter-channel correlation implicitly through shared weights. The resulting per-sample shape is $Z_{CI} \in \mathbb{R}^{T_{in}}$.

Sequence Formatting: After the CI, the look-back windows are formatted as text using a fixed decimal representation (Gruver et al., 2023). First each value is scaled (e.g. 10^5, Decimal Precision Sensitivity in Appendix H), and then converted from float to integer (Figure 1).If the input is *0.78, 0.985, 0.36581, 0.486, 0.758* it is transformed into *78000, 98500, 36581, 48600, 75800*. This eliminates the floats while preserving numerical precision up to five decimal places. By adopting this scaled representation, the input space is made more compact and stable, thus, reducing perturbations introduced to the embedding model and promoting more consistent downstream learning behavior.

Timestamp Formatting: The timestamp is formatted to include *month/day/weekday/hour/minute* **Figure 1**. For instance, if the timestamp is *"Date: 15-Sep Time: 09h25"*, it will be formatted as *9/15/3/9/25* removing distraction words (Date:, Time:) and creating a dense pure data that the model's attention mechanism can process without distractions.

Concatenation: The formatted sequence data are concatenated with the first timestamp of the sequence. We found that including all timestamps produces approximately the same results as including only the first. Including all timestamps increases the number of tokens in the sequence, leading to truncation if it exceeds the model's token limit (Timestamp Effect in Appendix I). Therefore, we decide to include only the first, providing a temporal context for the embedding and avoiding longer input sequences for the model.

3.2 Sequence Embedding

An embedding model usually is composed of several stacked transformer layers (Reimers & Gurevych, 2019; Karpukhin et al., 2020; Gao et al., 2021; Wang et al., 2022). Instead of using all the stacked layers to generate the dense embeddings, we perform layer pruning by selecting one layer and using its output to produce an embedding representation. The formatted sequences from the previous step are fed directly to the selected layer. The input initially is $Z_{formatted}$, representing the entire look-back window and the first timestamp as a single formatted string. The embedding model encode this via mean pooling over token representations, producing a single vector $Z_{embedding} \in \mathbb{R}^D$, $Z_{embedding} = PrunedEmbeddingModel(Z_{formatted})$, where D is the embedding model hidden size. The embedding model weights are frozen throughout training and no gradient updates are applied here.

3.3 Raw Sequences Residual

In this step, in addition to the embedding produced, we introduce raw sequence values Z_{CI} Figure 1. We perform point-wise addition between the raw values and the embedding vectors using zero padding to match the dimensions, this addition proved to be effective in improving model performance and ensures the original raw series information is not discarded. We can see it from the perspective as skip-connection between the input data and output of the embedding model: $Z_{emb} = PrunedEmbeddingModel(Z_{formatted}) + Z_{CI}$.

3.4 Forecasting Head

The vectors produced from the embedding space are packed with information about the time series sequences, from trends to seasonality. Hence the need for a forecasting head that is able to perform feature extraction and transform these features to prediction values. we propose CNN followed by a

simple linear output head. CNN was built to extract spatial features (Krizhevsky et al., 2012) from the data and proved to be effective (CNN vs Linear in Appendix G).

3.4.1 CONVOLUTION NEURAL NETWORKS (CNN)

The **CNN** in this work is composed of three consecutive convolution layers (see **Figure1**).
Expansion Layer: This layer is used to expand the channel dimension to C_{hidden} channels, $D_{exp} = D$ because the Kernel size $ks = 1$, $stride = 1$, and $padding = 0$, where D the input dimension, the values for each channel are calculated as: $u_c(n) = W_{\text{exp}}[c]\, z(n) + b_{\text{exp}}[c]$ where $W_{\text{exp}} \in \mathbb{R}^{C_{hidden} \times 1}$, z - input, $u_c(n)$ - value for point n in the output channel c after the expansion and b_{exp} - bias.
Feature Extraction Layer: This layer operates over the Z_{emb} vector and acts as feature extractor keeping the same number of channels C_{hidden} from the previous layer, the kernel size $ks > 1$, $stride = 1$, also the number of groups used is $g = C_{hidden}$ allowing the model to process each channel independently. For each channel c and for each position n the value is calculated as per below:

$$padding = \frac{(ks-1)}{2}, D_{ext} = \left\lfloor \frac{D_{exp} - 1}{stride} + 1 \right\rfloor$$

$$v_c(n) = \sum_{k=0}^{K-1} d_c[k]\, u_c(n+k) + b_{\text{ext}}[c], \quad d_c \in \mathbb{R}^K$$

Where $v_c(n)$ - the value for point n in the output channel c after the feature extraction , d_c - kernel weights for channel c, and b_{ext} - bias. The model performs direct rather than autoregressive forecasting, the non-causal convolution operates strictly within a fixed, fully observed historical look-back window, serving as a feature extractor without causing data leakage from the future prediction horizon.
Reduction Layer: This third layer is for processing and reducing the number of output channels and therefore the output head size, $D_{out} = D_{ext}$ because the Kernel size $ks = 1$, $stride = 1$, and $padding = 0$.

$$z_{cnn}(n) = \sum_{c=1}^{C_{\text{hidden}}} W_{\text{red}}[c]\, v_c(n) + b_{\text{red}}, W_{\text{red}} \in \mathbb{R}^{1 \times C_{\text{hidden}}}, Z_{cnn} \in \mathbb{R}^{1 \times D_{\text{out}}}$$

The number of output channels of the CNN $C_{out} = 1$ and b_{red} is the bias.

3.4.2 OUTPUT HEAD

The output head is a linear projection of flattened CNN output to the desired forecasting horizon. $\hat{Y} = RevIN(((Z_{cnn} + Z_{CI})W_1^T + b_1)W_2^T + b_2)$, $W_1 \in \mathbb{R}^{h \times C_{out} \cdot D_{out}}, W_2 \in \mathbb{R}^{T_{out} \times h}, h = 2T_{out}$ We discovered that using any activation function between the two linear layers resulted in a performance decrease. This is because most standard activations restrict or eliminate negative values making it difficult for the network to predict the target with $\mu = 0, \sigma = 1$. Therefore we decided to keep it linear (Details Activation Function Appendix).

4 EXPERIMENTS

4.1 DATASETS

To fully test the usefulness of text embedding in TSF, we tested our model on five benchmark datasets Table 1. We evaluated the model on these datasets to ensure an extensive benchmark across varied sampling frequencies, high-dimensional features and challenging real-world patterns. These are publicly available datasets used for benchmarking (Wu et al., 2023a), (Wang et al., 2026) [1].

4.2 EVALUATION METRICS

The main metrics used in the evaluation of time series long-term forecasting are Mean Square Error (MSE) and Mean Absolute Error (MAE). MSE and MAE are utilized as the primary evaluation

[1] https://huggingface.co/datasets/thuml/Time-Series-Library/tree/main

Table 1: 5 Benchmark datasets used for evaluation.

Dataset	Features	Timesteps	Granularity
Weather	21	52,696	10 min
ETTh1 & ETTh2	7	17,420	1 h
ETTm1 & ETTm2	7	69,680	15 min

metrics because they serve as the universal benchmark standard in time-series literature, offering a balanced assessment of both average error and sensitivity to large outliers. We use Average ranking which scores models from 1st to last place on every individual test, calculating their mean rank to show which model is the most consistent performer across all datasets and horizons. The reported results are for a fixed seed, however we run the model using different seeds (details in Confidence Intervals Appendix) and we calculated the confidence interval Table 4.

4.3 Baseline Models

For benchmark and evaluation, we compare against seven competitive baselines: 1) **One Fits All** (Zhou et al., 2023) uses CI and patching concepts while relying on GPT-2 as backbone maintaining its pretrained weights. 2) **Time-LLM** (Jin et al., 2024) patches and reprograms the time series data and uses prompting as prefix concept to guide LLMs forecasting, the model achieved an excellent performance on few shot learning while keeping the pretrained weights of the backbone model. 3) **Dlinear** (Zeng et al., 2023) introduced a simple model to challenge the use of transformers in TSF, it outperformed several complex models. 4)**PatchTST** (Nie et al., 2023) introduced CI and Patching concepts, it reduced the complexity of the attention mechanisms outperforming the standard transformer based models and challenging DLinear. 5) **TEMPO** (Cao et al., 2024) adapts GPT models and uses time series decomposition with prompt adaptation, it excelled in zero-shot scenarios. 6) **TEST** (Sun et al., 2024) aligns time series data to the LLM embedding enabling forecasting. 7) **LLM4TS** (Chang et al., 2025) uses two stage finetuning to adapt LLMs to time series data. These baselines were chosen to represent traditional (PatchTST, DLinear) and the current LLM-based models comparisons.

4.4 Implementation details

We use the experimental setup (train/test/val splits) from LLM4TS (Chang et al., 2025) for comparison consistency [2]. We evaluate our model performance on few shot scenarios using 10 % of the training data. Then we explore the model performance on full shot scenario for balanced evaluation. The test and validation data are similar for both evaluation scenarios. We set $T_{in} = 336$ for fair comparison . The embedding model used in the experiment is *all-MiniLM-L6-v2* with a hidden size of *384*, $C_{hidden} = 3$. We pruned the embedding model and we used the first layer to produce the dense embeddings (Appendix B.3). These hyperparameters were selected based on experimental results. We trained our model using the AdamW optimizer with a Cosine Annealing with Warm Restarts learning rate schedule, in addition to early stopping once the loss is stabilized. All code and hyperparameter configurations will be publicly available after acceptance.

5 Results and Analysis

5.1 Few-shot learning - 10% training data

Across Weather and the ETT benchmarks Table2, **PTEF** achieves the best overall average MSE rank of 2.0 and the best MAE rank of 2.2, showing consistent performance under the 10% training-data setting. PTEF obtains the lowest MSE and MAE on **ETTm1**, the lowest MAE and a tied-best MSE on **Weather**, and remains close to the best methods on **ETTm2**. On the hour-level datasets, **ETTh1** and **ETTh2**, PTEF remains competitive, although TEMPO and LLM4TS achieve better results. Overall, PTEF provides a strong accuracy model size trade-off and outperforms all state-of-the-art methods considered in our comparison in terms of average rank.

[2] https://github.com/blacksnail789521/LLM4TS

Table 2: **Few-shot long-term forecasting using 10% of the training data:** Results on Weather and ETT benchmarks (ETTh1, ETTh2, ETTm1, ETTm2). For each dataset, MSE and MAE are averaged across the four prediction horizons $T_{\text{out}} \in \{96, 192, 336, 720\}$ (lower is better). **Avg. Rank** computed across all datasets and horizons. The best results in ↓red second-best in blue. Results for competing baselines are taken from LLM4TS (Chang et al., 2025). TEMPO* is evaluated under a zero-shot setting as it is a prompt-based method.

Models	**PTEF (ours)**		LLM4TS		GPT4TS		DLinear		PatchTST		Time-LLM		TEMPO*		TEST	
Dataset	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	↓**0.234**	↓**0.269**	0.235	**0.270**	0.238	0.275	0.241	0.283	0.242	0.279	↓**0.234**	0.273	0.282	0.316	0.238	0.272
ETTh1	**0.523**	**0.493**	0.525	**0.493**	0.590	0.525	0.691	0.600	0.633	0.542	0.555	0.522	↓**0.428**	↓**0.427**	0.582	0.525
ETTh2	0.390	0.416	**0.366**	0.407	0.397	0.421	0.605	0.538	0.415	0.431	0.371	↓**0.394**	↓**0.361**	**0.398**	0.400	0.432
ETTm1	↓**0.396**	↓**0.405**	0.408	**0.413**	0.464	0.441	0.411	0.429	0.501	0.466	**0.404**	0.427	0.501	0.458	0.460	0.443
ETTm2	**0.277**	0.325	↓**0.276**	**0.324**	0.293	0.335	0.316	0.368	0.296	0.343	**0.277**	↓**0.323**	0.281	0.328	0.337	0.331
Avg. Rank	↓**2.000**	↓**2.200**	**2.400**	↓**2.200**	5.200	5.200	6.600	7.000	6.800	6.800	**2.400**	2.600	4.200	4.400	5.600	5.200

Table 3: **Full-shot Long-term forecasting:** Results on Weather and ETT benchmarks (ETTh1, ETTh2, ETTm1, ETTm2). For each dataset, MSE and MAE are averaged across the four prediction horizons $T_{\text{out}} \in \{96, 192, 336, 720\}$ (lower is better). **Avg. Rank** computed across all datasets and horizons. The best results in ↓red second-best in blue. Results for competing baselines are taken from LLM4TS (Chang et al., 2025).

Models	**PTEF (ours)**		LLM4TS		GPT4TS		DLinear		PatchTST		Time-LLM		TEMPO*		TEST	
Dataset	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	0.234	0.270	↓**0.223**	**0.260**	0.237	0.271	0.249	0.300	**0.226**	0.264	**0.226**	↓**0.258**	0.282	0.316	0.229	0.269
ETTh1	**0.408**	**0.422**	↓**0.404**	↓**0.418**	0.428	0.426	0.423	0.437	0.413	0.431	0.408	0.424	0.428	0.427	0.414	0.432
ETTh2	0.338	0.387	0.333	0.383	0.355	0.395	0.431	0.447	↓**0.330**	↓**0.379**	0.334	0.383	0.361	0.398	**0.331**	**0.380**
ETTm1	0.353	**0.376**	**0.343**	0.378	0.352	0.383	0.357	0.379	0.351	0.381	↓**0.329**	↓**0.372**	0.501	0.458	0.353	0.382
ETTm2	0.256	0.315	↓**0.251**	**0.313**	0.267	0.326	0.267	0.334	0.255	0.315	↓**0.251**	0.314	0.281	0.328	0.279	↓**0.291**
Avg. Rank	4.200	3.600	↓**1.600**	↓**2.200**	5.600	5.800	6.600	7.000	2.600	3.800	**2.000**	↓**2.200**	7.600	7.000	4.600	4.000

Table 4: ETTm1 results reported as mean and 95% confidence interval across different seeds.

Dataset	T_{out}	**MSE Avg.**	**MSE (95% CI)**	**MAE Avg.**	**MAE (95% CI)**
ETTm1	96	0.344	[0.343, 0.344]	0.375	[0.374, 0.376]
	192	0.371	[0.369, 0.374]	0.391	[0.389, 0.392]
	336	0.404	[0.402, 0.407]	0.410	[0.409, 0.411]
	720	0.464	[0.461, 0.468]	0.447	[0.445, 0.449]

Table 5: Impact of CNN hidden channels on ETTh1 using 100% training data.

Models		$C_{\text{hidden}} = 3$		$C_{\text{hidden}} = 12$		$C_{\text{hidden}} = 24$	
Dataset	T_{out}	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	96	0.366	0.391	↓**0.363**	↓**0.390**	↓**0.363**	↓**0.390**
	192	**0.401**	↓**0.412**	↓**0.400**	↓**0.412**	**0.401**	↓**0.412**
	336	0.427	↓**0.429**	↓**0.425**	↓**0.429**	**0.426**	0.430
	720	0.439	0.457	**0.435**	**0.455**	↓**0.434**	↓**0.452**

5.2 Full-shot Learning

In the full-shot setting, **PTEF** remains competitive across the Weather and ETT benchmarks Table3, with average ranks of 4.2 for MSE and 3.6 for MAE. PTEF achieves the second-best MAE of 0.376 on **ETTm1** and matches PatchTST with an MAE of 0.315 on **ETTm2**. PTEF also ties with Time-LLM for the second-best MSE of 0.408 and had the second best MAE of 0.422 on **ETTh1**. Compared with the few-shot setting, PTEF's advantage is smaller in the full-shot scenario. This may be due to its compact model size, which can limit its ability to benefit from larger training data compared with larger LLM-based models such as Time-LLM and LLM4TS. Increasing the number of CNN hidden channels improves full-shot performance Table 5. Despite this capacity limitation, PTEF achieves strong overall ranks (4.2 MSE / 3.6 MAE), outperforming classical models such as DLinear (Zeng et al., 2023), TEMPO (Cao et al., 2024), TEST (Sun et al., 2024), and One Fits ALL (Zhou et al., 2023), and remains competitive with PatchTST (Nie et al., 2023). These results show that PTEF provides a good accuracy–model size trade-off.

5.3 Trainable Parameters and Inference Cost

Models with a low number of trainable parameters provide clear advantages in terms of efficiency, scalability, and practical deployment, especially when strong performance is maintained. LLM4TS has already established a clear advantage in this perspective over other architectures like PatchTST and FEDformer. PTEF demonstrates superior efficiency using only 0.094M trainable parameters

(0.68%) out of 13.8M total, compared to 3.4M (4%) out of 85M for LLM4TS Table 6.
This efficiency is largely due to the use of a pre-trained embedding component in PTEF, which is applied as a **single forward pass** without requiring gradient updates. As a result, the embedding contributes significantly to the model's representational power without increasing training complexity or cost. In Table 7 we detailed the training time using $T_{out} = 96$, ETTh2 dataset, and NVIDIA RTX A6000. We noticed that after the embedding is performed, the training time is 24 times faster and inference 28 times faster. On the other hand, the full pipeline requires 3.54 GMACs, with the embedding model accounting for 3.52 GMACs (99.44% of the total cost). The forecasting head is lightweight, adding only 0.0116 GMACs (0.56%). compared with LLM4TS (Chang et al., 2025) at $\approx$ 12.77 GMACs, PTEF is 3.6 times less expensive due the layer pruning performed and its light forecasting head (details in Inference Cost Appendix K).
Overall, PTEF achieves a strong balance between performance and efficiency by combining a lightweight trainable core with a fixed embedding stage. This leads to reduced memory usage, faster training and inference, and easier deployment, while remaining competitive with larger partially trainable models such as LLM4TS.

Table 6: Model size comparison in terms of trainable parameters, total parameters, and trainable parameter percentage. The PTEF count includes the pruned embedding model.

Model	Trainable	Total	Percentage
LLM4TS	3.4M	85M	4%
PTEF	0.094M	13.8M	0.68%

Table 7: PTEF training and inference time per batch (seconds) on ETTh2, $T_{\text{out}} = 96$. Longer first epoch includes embedding, subsequent epochs train only the forecasting head.

	Training	Test
1st epoch	0.099	0.084
2nd epoch	0.004	0.003

6 Ablation studies

6.1 Effectiveness of Text Embedding

To assess the impact of text embedding on model performance, we removed the embedding stage, and then we compared the results with and without text embedding. For the without text embedding experiment, we adopted the same model components except for the timestamp embedding. We used trainable embedding tables for each time component *minute, hour, weekday, day, and month* (Appendix C.2). Then the time embedding vector is concatenated with the time series data as we found the concatenation provides better results. We analyzed the results of both scenarios on 3 different datasets Table 8. We noticed clearly that text embedding is the main driver behind the performance, without it the model acts approximately as a linear model.

6.2 Generalization over different Embedding Models

We tested PTEF with different text embedding models with their full layers including *all-MiniLM-L6-v2*, *bge-small-en-v1.5* (Xiao et al., 2024), and *all-mpnet-base-v2* (Song et al., 2020; Reimers & Gurevych, 2019), their embedding sizes are *384*, *384*, and *768* consecutively (Models' details in Appendix B.2). We observed that PTEF consistently exhibits a sate of the art performance compared to the other models in terms of average ranking on the 5 different datasets Table 9, while keeping the model size and trainable parameters to a fraction of the current SOTA models. *all-MiniLM-L6-v2* and *bge-small-en-v1.5* achieved an average ranking of 2.450, and *all-mpnet-base-v2* achieved 2.500 average ranking in term of MSE surpassing the previous SOTA Models while only trained on 10 percent of the data. Among these text embedding models we noticed that higher embedding size leads to increase in trainable parameters and sometimes a negative impact on performance, if we compare MSE scores of *all-MiniLM-L6-v2* and *all-mpnet-base-v2*, we observe that the model with the small embedding size performed better.

6.3 Layer Pruning

To find the best layer for generating dense embeddings and check wether layer pruning works across different models, we evaluated each layer of the three embedding models used in Section 6.2 on sev-

Table 8: PTEF performance across datasets and prediction horizons, with and without text embeddings. For with Text Embedding no layer drop performed.

Models		PTEF / Embd		PTEF / No Embd	
Dataset	T_{out}	MSE	MAE	MSE	MAE
Weather	96	↓0.156	↓0.206	0.163	0.215
	192	↓0.199	↓0.244	0.212	0.258
	336	↓0.250	↓0.283	0.263	0.298
	720	↓0.325	↓0.336	0.335	0.345
ETTh1	96	↓0.419	↓0.427	0.440	0.448
	192	0.512	0.487	↓0.497	↓0.480
	336	↓0.539	↓0.506	0.547	0.512
	720	↓0.635	↓0.563	0.663	0.576
ETTm1	96	↓0.343	↓0.374	0.361	0.389
	192	↓0.372	↓0.391	0.414	0.420
	336	↓0.405	↓0.410	0.430	0.431
	720	↓0.462	↓0.445	0.470	0.452

Table 9: Effect of embedding models on forecasting performance across datasets and prediction horizons using 10% training data.

Models		MiniLM-L6-v2		BGE-small-v1.5		MPNet-base-v2	
Dataset	T_{out}	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	0.156	0.206	0.156	0.206	0.157	0.208
	192	0.199	0.244	0.199	0.244	0.201	0.246
	336	0.250	0.283	0.250	0.283	0.252	0.285
	720	0.325	0.336	0.325	0.335	0.325	0.337
ETTh1	96	0.419	0.427	0.419	0.428	0.418	0.426
	192	0.512	0.487	0.495	0.476	0.476	0.466
	336	0.539	0.506	0.546	0.508	0.540	0.505
	720	0.635	0.563	0.656	0.573	0.671	0.579
ETTh2	96	0.315	0.365	0.315	0.362	0.316	0.367
	192	0.397	0.408	0.395	0.407	0.393	0.406
	336	0.404	0.431	0.405	0.426	0.405	0.430
	720	0.458	0.469	0.456	0.469	0.459	0.470
ETTm1	96	0.343	0.374	0.344	0.374	0.347	0.377
	192	0.372	0.391	0.371	0.392	0.373	0.393
	336	0.405	0.410	0.403	0.409	0.405	0.411
	720	0.462	0.445	0.464	0.448	0.466	0.449
ETTm2	96	0.178	0.260	0.177	0.261	0.179	0.261
	192	0.234	0.301	0.237	0.301	0.236	0.301
	336	0.288	0.337	0.289	0.338	0.291	0.339
	720	0.399	0.398	0.402	0.400	0.402	0.399

eral datasets. The results (Figures 2, 3, 4 Appendix B.3) show that the best layer changes depending on the model and dataset. Using a single layer often gives results close to or better than those of the full encoder. In some cases the full encoder performs worse than using only one layer. these results show that layers can be dropped from pretrained text embedding models during inference without extra training and without a large decrease in performance.

7 Conclusion

This work introduce PTEF, a novel and efficient architecture for time series forecasting that leverages pruned pre-trained text embeddings as a core component. By decoupling representation learning from forecasting and using the embedding stage as a single forward pass, PTEF significantly reduces the number of trainable parameters while maintaining strong predictive performance. Extensive experiments across multiple benchmark datasets demonstrate that PTEF remains competitive and in some cases outperforms existing methods, particularly in few-shot scenarios. Furthermore PTEF becomes more efficient by dropping layers in the text embedding stage without a major loss in performance while reducing its latency. Additionally, our analysis shows that the choice of embedding model has limited impact on performance, highlighting the robustness of the approach. Overall, PTEF provides an effective and scalable alternative to larger models, offering a favorable balance between efficiency, accuracy, and practical deployment.

8 Limitations:

Despite its efficiency and strong forecasting performance, PTEF still has some limitations. Its fixed embedding stage reduces training cost but limits dataset-specific adaptation, which may affect performance in some settings. Moreover, although PTEF has a compact number of trainable parameters, its end-to-end MAC cost remains considerably higher compared to linear models due to the text embedding component even after pruning. Additionally, the small embedding models used have a token limit which prohibit us from profiting from larger look-back windows. Future work should focus on improving the tokenization and efficiency of the embedding stage while increasing model capacity without compromising PTEF's lightweight design.

9 AI USE STATEMENT

The authors used generative AI tools to assist with LaTeX formatting, code snippets for manuscript preparation, and language polishing. All scientific content, experimental results, analyses, claims, and final text were reviewed and verified by the authors.

REFERENCES

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

Defu Cao, Furong Jia, Sercan O Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. TEMPO: Prompt-based generative pre-trained transformer for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=YH5w12OUuU.

Ching Chang, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. Llm4ts: Aligning pre-trained llms as data-efficient time-series forecasters. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 16(3), 2025. doi: 10.1145/3719207.

Jialin Chen, Jan Eric Lenssen, Aosong Feng, Weihua Hu, Matthias Fey, Leandros Tassiulas, Jure Leskovec, and Rex Ying. From similarity to superiority: Channel clustering for time series forecasting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=MDgn9aazo0.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6894–6910, 2021.

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. URL https://openreview.net/forum?id=md68e8iZK1.

Lu Han, Han-Jia Ye, and De-Chuan Zhan. The capacity and robustness trade-off: Revisiting the channel independent strategy for multivariate time series forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. What matters in transformers? not all attention is needed, 2024. URL https://arxiv.org/abs/2406.15786.

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. URL https://openreview.net/forum?id=Unb5CVPtae.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Ledell, Ali Zaidi, Devendra Bajaj, Kai-Wei Chang, Danqi Chen, Wen-tau Yih, Kristin Wang, and Luke Zettlemoyer. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020.

Rachneet Kaur, Zhen Zeng, Tucker Balch, and Manuela Veloso. LETS-C: Leveraging text embedding for time series classification. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32365–32399, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1557. URL https://aclanthology.org/2025.acl-long.1557/.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=cGDAkQo1C0p.

Siva Rama Krishna Kottapalli, Karthik Hubli, Sandeep Chandrashekhara, Garima Jain, Sunayana Hubli, Gayathri Botla, and Ramesh Doddaiah. Foundation models for time series: A survey, 2025. URL https://arxiv.org/abs/2504.04011.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=JePfAI8fah.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1035. URL https://aclanthology.org/2025.findings-acl.1035/.

Reza Nematirad, Anil Pahwa, and Balasubramaniam Natarajan. Are data embeddings effective in time series forecasting?, 2025. URL https://arxiv.org/abs/2505.20716.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Jbdc0vTOcol.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. Poor man's BERT: smaller and faster transformer models. *CoRR*, abs/2004.03844, 2020. URL https://arxiv.org/abs/2004.03844.

Christopher A. Sims. Macroeconomics and reality. *Econometrica*, 48(1):1–48, 1980. doi: 10.2307/1912017.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 16857–16867. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c3a690be93aa602ee2dc0ccab5b7b67e-Paper.pdf.

Chenxi Sun, Hongyan Li, Yaliang Li, and Shenda Hong. TEST: Text prototype aligned embedding to activate LLM's ability for time series. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Tuh4nZVb0g.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *ArXiv*, abs/2212.03533, 2022. URL https://api.semanticscholar.org/CorpusID:254366618.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Yong Liu, Chen Wang, Mingsheng Long, and Jianmin Wang. Deep time series models: A comprehensive survey and benchmark, 2026. URL https://arxiv.org/abs/2407.13278.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=J4gRj6d5Qm.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *International Conference on Learning Representations*, 2023a.

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*, 2023b. URL https://openreview.net/forum?id=ju_Uqw384Oq.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 641–649, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657878. URL https://doi.org/10.1145/3626772.3657878.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11121–11128, Jun. 2023. doi: 10.1609/aaai.v37i9.26317. URL https://ojs.aaai.org/index.php/AAAI/article/view/26317.

Anhao Zhao, Fanghua Ye, Yingqi Fan, Junlong Tong, Jing Xiong, Zhiwei Fei, Hui Su, and Xiaoyu Shen. SkipGPT: Each token is one of a kind. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=d7v2iUSa9s.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):11106–11115, May 2021. doi: 10.1609/aaai.v35i12.17325. URL https://ojs.aaai.org/index.php/AAAI/article/view/17325.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27268–27286. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/zhou22g.html.

Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. One fits all: Power general time series analysis by pretrained LM. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=gMS6FVZvmF.

A Experiments

A.1 Datasets

Benchmark datasets details:

Weather : This dataset contains meteorological measurements from the Max Planck Institute for Biogeochemistry. It includes 21 indicators such as air temperature, humidity, and atmospheric pressure. It is widely used for multivariate forecasting tasks (Wu et al., 2021).

ETTh1 & ETTh2 (Electricity Transformer Temperature): The ETT (Zhou et al., 2021) datasets are the primary benchmarks for long-horizon forecasting. They track data from electricity transformers in two different counties in China.

ETTm1 & ETTm2: These are the higher-resolution versions of the ETT (Zhou et al., 2021) data. Because of the finer granularity, these datasets have 4x the number of timesteps compared to the ETTh variants, making them useful for testing high-frequency pattern recognition.

A.2 Evaluation Metrics

The Mean Square Error (MSE) is expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The Mean Absolute Error (MAE) is expressed as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Where y_i and $\hat{y}_i$ are the true and estimated values for sample i respectively.

B Ablation Studies

B.1 Effectiveness of Text Embedding

PTEF with text embedding (PTEF/Embd) significantly outperforms PTEF without text embedding (PTEF / No Embd), achieving a much better average rank across 3 benchamrk datasets and horizons Table 10:
PTEF / Embd: Avg. Rank of 2.083 for both MSE and MAE.
PTEF / No Embd: Avg. Rank of 4.500 for MSE and 4.083 for MAE.
Thus, integrating embeddings provides the model a consistent boost in forecasting accuracy across almost all scenarios, demonstrating the effectiveness of pretrained embedding models.

Table 10: Performance comparison (MSE and MAE) across multiple datasets and prediction horizons for **PTEF with and without Text Embedding** and the baseline models. For with Text Embedding no layer drop performed. Best results are highlighted red and second best in blue.

Dataset	T_{out}	PTEF / Embd		PTEF / No Embd		LLM4TS		DLinear		Time-LLM		TEMPO*		TEST	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	**↓0.156**	**↓0.206**	0.163	0.215	**0.158**	**0.207**	0.171	0.224	0.161	0.210	0.211	0.254	0.163	0.213
	192	**↓0.199**	**↓0.244**	0.212	0.258	**0.204**	0.249	0.215	0.263	**0.204**	**0.248**	0.254	0.298	0.230	0.263
	336	**↓0.250**	**0.283**	0.263	0.298	0.254	0.288	0.258	0.299	0.261	0.302	0.292	0.332	0.258	**↓0.282**
	720	0.325	0.336	0.335	0.345	0.322	0.336	0.320	0.346	**0.309**	**0.332**	0.370	0.379	**↓0.301**	**↓0.328**
ETTh1	96	0.419	**0.427**	0.440	0.448	**0.417**	0.432	0.492	0.495	0.448	0.460	**↓0.400**	**↓0.406**	0.455	0.457
	192	0.512	0.487	0.497	0.480	**0.469**	**0.468**	0.565	0.538	0.484	0.483	**↓0.426**	**↓0.419**	0.572	0.519
	336	0.539	0.506	0.547	0.512	**0.505**	**0.499**	0.721	0.622	0.589	0.540	**↓0.441**	**↓0.430**	0.578	0.531
	720	**0.635**	**0.563**	0.663	0.576	0.708	0.572	0.896	0.743	0.700	0.604	**↓0.443**	**↓0.451**	0.723	0.594
ETTm1	96	**↓0.343**	**↓0.374**	0.361	0.389	0.360	**0.388**	0.352	0.392	**0.346**	**0.388**	0.438	0.424	0.392	0.401
	192	**↓0.372**	**↓0.391**	0.414	0.420	0.386	**0.401**	0.382	0.412	**0.373**	0.416	0.461	0.432	0.423	0.426
	336	**↓0.405**	**↓0.410**	0.430	0.431	0.415	**0.417**	0.419	0.434	**0.413**	0.426	0.515	0.467	0.471	0.444
	720	**↓0.462**	**0.445**	0.470	0.452	**0.470**	**↓0.445**	0.490	0.477	0.485	0.476	0.591	0.509	0.552	0.501

B.2 Generalization across various Embedding Models

To evaluate the generalization of text embeddings, we experimented with three different embedding models, their summary in Table 11. The models are ranked using Borda, for each dataset, models receive points based on their placement (higher rank = more points), and the model with the highest total points across all datasets wins the top Borda rank.

B.2.1 all-MiniLM-L6-v2

The is one of the very small embedding models that maps sentences and paragraphs to *384* dense embedding vector, it can be used for semantic search or clustering tasks. We tested with this em-

Table 11: Comparison of embedding models based on MTEB ranking, embedding dimensions, and maximum token length

Rank (Borda)	Model	Embedding Dimensions	Max Tokens
115	`bge-small-en-v1.5`	384	512
132	`all-mpnet-base-v2`	768	384
145	`all-MiniLM-L6-v2`	384	256

bedding model across 5 benchmark datasets, despite the small size and max token length, the model exhibited SOTA performance on some of these datasets Table 9.

B.2.2 BGE-SMALL-EN-V1.5

is a lightweight English text embedding model, that converts text into *384*-dimensional dense vectors for semantic search, retrieval (RAG), clustering, and similarity matching (Xiao et al., 2024)

B.2.3 ALL-MPNET-BASE-V2

This embedding model was built on mpnet-base (Song et al., 2020) , fine-tuned on the 1B sentence-pair dataset with a contrastive learning objective, and using the training approach from (Reimers & Gurevych, 2019). The model hidden size is *768*.

B.3 Pruned Embedding Model

Layer pruning is a model compression technique that reduces the computational cost by removing selected layers while preserving most of the model's performance. The main idea is that not all transformer layers contribute equally to the final representation, and some layers can be skipped or removed during inference with only a small accuracy degradation. By evaluating the performance of individual layers, layer pruning enables a more efficient model with fewer parameters, lower memory usage, and reduced inference latency. In this section we evaluated performance of each layer for three different embedding models Appendix B.2. The results in Figures 2, 3, 4 demonstrated that representation produced by one layer achieves the same and sometimes better performance.

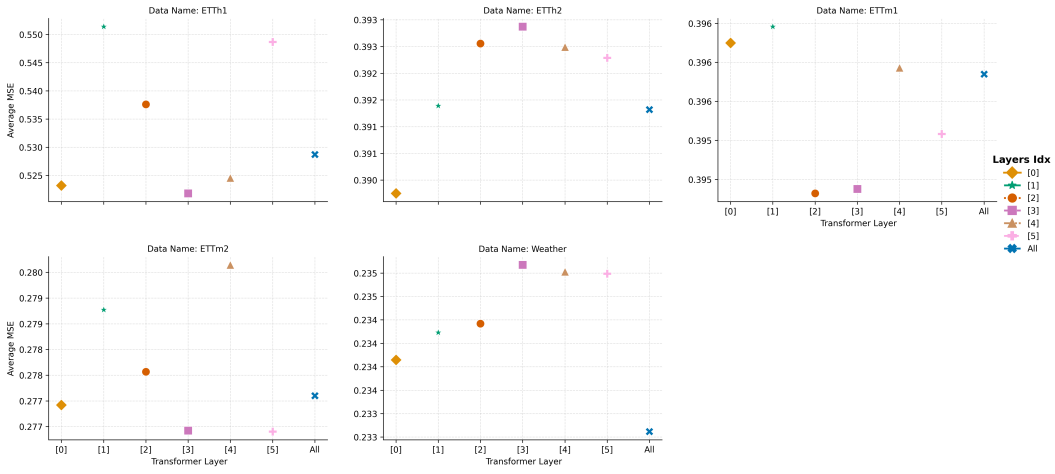

Figure 2: **all-MiniLM-L6-v2:** Comparing performance of different layers and the full model to generate dense embeddings. MSE average across $T_{\text{out}} \in \{96, 192, 336, 720\}$.

C Ideas that failed

In this section we mention ideas that did not work to prevent other from making the same mistakes.

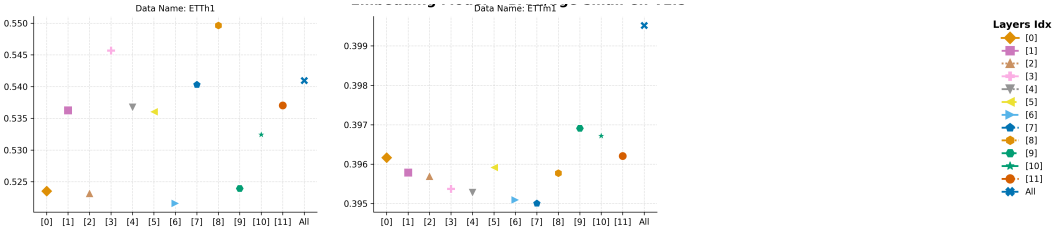

Figure 3: **bge-small-en-v1.5:** Comparing performance of different layers (x-axis) and the full model to generate dense embeddings. MSE average (y-axis) across $T_{\text{out}} \in \{96, 192, 336, 720\}$.

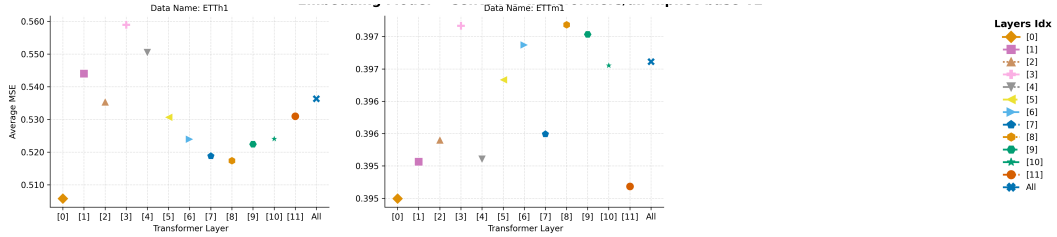

Figure 4: **all-mpnet-base-v2:** Comparing performance of different layers and the full model to generate dense embeddings. MSE average across $T_{\text{out}} \in \{96, 192, 336, 720\}$.

C.1 Patching Before Text Embedding

We tried using patching concept (Nie et al., 2023) before performing text embedding and then passing the embedded patches through tiny transformer **Figure 5**, but we found performance decrease. The transformer added unnecessary capacity with increase in the emebedding cost.

C.2 Trainable Time Embeddings

Instead of the of including the timestamp in the sequence before text embedding, we tried using trainable time embedding equation 1 and add or concatenate it with the output of the embedding model. We found that the performance dropped, the text embedding models are already trained to understand time, therefore, no need to add additional trainable parameters. When testing the without text embedding scenario, we have to introduce for each sequence a timestamp, to capture trends and periodicity, as the sequence is composed of multiple sample points we use the time stamp of the first sample, and embedded it using the same concept.

$$Z_{time} = E_{month} + E_{day} + E_{weekday} + E_{hour} + E_{minute} \tag{1}$$

E_x is retrieved from the embedding table of x

D Activation Function

The Output head adopted for the architecture has two consecutive linear layers without any activation function, we discovered that in the case of a normalized target using an activation will eliminate or restrict the negative values making it hard for the network to map the predictions to normalized values with $\mu = 0$ and $\sigma = 1$ Table 12.

E Spaced Digit

Adding space between digits might impact forecasting performance negatively Table 13, spacing digits achieved a lower average score compared to leaving no spaces. The main drawback with adding spaces is that it makes the text much longer. This forces the model to use more tokens, which can cause the data to get truncated if it exceeds the model's token limit.

Table 12: Comparison of different activation functions on the ETTm1 dataset using 10% training data. Results using the full embedding model.

Models		No Activation		ELU		Relu	
Dataset	T_{out}	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	96	↓ **0.343**	↓ **0.374**	**0.392**	**0.406**	0.429	0.428
	192	↓ **0.372**	↓ **0.391**	**0.410**	**0.417**	0.493	0.465
	336	↓ **0.405**	↓ **0.410**	**0.431**	**0.432**	0.482	0.458
	720	↓ **0.462**	↓ **0.445**	**0.496**	**0.468**	0.568	0.503
Avg. Rank		↓ **1.000**	↓ **1.000**	**2.000**	**2.000**	3.000	3.000

Table 13: Performance comparison between No Space and Spaced digits configurations. Results using the full embedding model.

Models		No Space		Spaced digits	
Dataset	T_{out}	MSE	MAE	MSE	MAE
Weather	96	↓ **0.156**	↓ **0.206**	**0.156**	**0.206**
	192	**0.199**	↓ **0.244**	↓ **0.199**	**0.245**
	336	↓ **0.250**	↓ **0.283**	**0.250**	**0.283**
	720	**0.325**	**0.336**	↓ **0.325**	↓ **0.336**
ETTh1	96	**0.419**	↓ **0.427**	↓ **0.418**	**0.429**
	192	**0.512**	**0.487**	↓ **0.511**	↓ **0.487**
	336	**0.539**	**0.506**	↓ **0.537**	↓ **0.505**
	720	↓ **0.635**	↓ **0.563**	**0.650**	**0.569**
ETTh2	96	↓ **0.315**	↓ **0.365**	**0.319**	**0.366**
	192	**0.397**	**0.408**	↓ **0.392**	↓ **0.406**
	336	↓ **0.404**	↓ **0.431**	**0.405**	**0.431**
	720	**0.458**	**0.469**	↓ **0.456**	↓ **0.468**
ETTm1	96	**0.343**	**0.374**	↓ **0.342**	↓ **0.373**
	192	**0.372**	**0.391**	↓ **0.371**	↓ **0.391**
	336	↓ **0.405**	↓ **0.410**	**0.406**	**0.410**
	720	↓ **0.462**	↓ **0.445**	**0.462**	**0.445**
ETTm2	96	↓ **0.178**	**0.260**	**0.178**	↓ **0.260**
	192	↓ **0.234**	**0.301**	**0.234**	↓ **0.301**
	336	↓ **0.288**	↓ **0.337**	**0.292**	**0.338**
	720	↓ **0.399**	↓ **0.398**	**0.403**	**0.399**
Avg. Rank		↓ **1.450**	↓ **1.450**	**1.550**	**1.550**

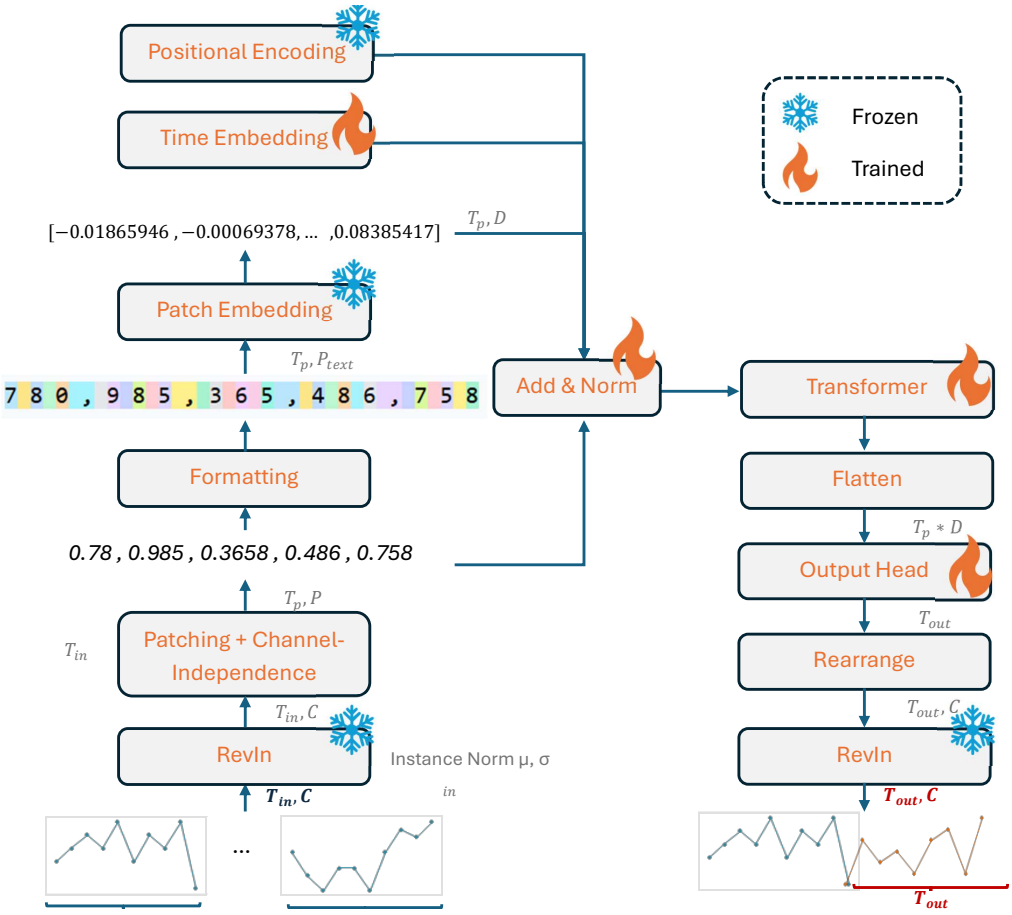

Figure 5: **Transformer Based Architecture with patching:** In this architecture we first normalize the time series data using reversible instance normalization, use Channel-independence and patching, format the patches into a text representation, generate embeddings for them . The embedding vectors are then combined with, the original time series, positional encoding, and time embedding using point-wise addition. Then a transformer followed by an linear projection is used to perform the final horizon prediction. During training, only the transformer and linear projection components are trained.

Table 14: Evaluation of the residual connection impact using ETTm1 dataset 10% training data. Results using the full embedding model.

Models		Residual Connection		No Residual Connection	
Dataset	$\mathbf{T_{out}}$	**MSE**	**MAE**	**MSE**	**MAE**
ETTm1	96	↓ **0.343**	↓ **0.374**	**0.344**	**0.375**
	192	↓ **0.372**	↓ **0.391**	**0.375**	**0.393**
	336	↓ **0.405**	↓ **0.410**	**0.409**	**0.412**
	720	↓ **0.462**	↓ **0.445**	**0.468**	**0.449**
Avg. Rank		↓ **1.000**	↓ **1.000**	**2.000**	**2.000**

F CNN Residual Connection

The raw time series data is added as a residual connection after the CNN, the experiments demonstrated that incorporating this residual connection strictly improves model performance across all evaluated forecasting horizons (T_{out}) on the ETTm1 dataset Table 14. With this addition, we miti-

gates effectively information loss, allowing the raw time-series data to bypass intermediate transformations.

G CNN VS LINEAR FORECASTING HEAD

In this experiment we compared the CNN forecasting head against simple linear one. We used 10% training data from ETTh1 and we removed the cnn from the architecture.
The model with CNN exhibited a superior performance compared to the model with a simple linear head Table 15.

Table 15: Comparison of PTEF with and without CNN using 10% training data of ETTh1. Results using the full embedding model.

Dataset	T_{out}	PTEF (ours)		PTEF without CNN	
		MSE	MAE	MSE	MAE
ETTh1	96	↓**0.419**	↓**0.427**	0.434	0.444
	192	↓**0.512**	↓**0.487**	0.517	0.492
	336	0.539	0.506	↓**0.538**	↓**0.505**
	720	↓**0.635**	↓**0.563**	0.643	0.566

H DECIMAL PRECISION SENSITIVITY

In the main architecture, we have the sequence formatting step, which converts normalized time series data into integers using a fixed decimal precision. in this section, we explored the model sensitivity to the different decimal precisions. During this experiment, we used 5 different benchmark datasets and tested with 10^2, 10^5, and 10^7.
The results showed that the model performance increases with the increase of the decimal precision Table 16, these results align with the concept that the more we increase the precision the more we reduce the similarity between data points, which helps the model extract different features and patterns. on the other hand, the performance improvement comes at a cost, the more we increase the decimal precision the longer the sequence will get, which may exceed the token limit of the embedding model.

I TIMESTAMP EFFECT

In this section we tested the impact of timestamps on the model performance, we experimented with the first timestamp only (main architecture), without timestamp, and including all timestamps in the formatted sequence.
The results demonstrated that the impact is minor between these three different scenarios Table 17. including only the first and all timestamps have approximately the same results, while not using any timestamps resulted in minor decrease in some cases.

J FULL EMBEDDING MODEL

In this present the results without pruning, the embedding model was used with its full layers.

J.1 FEW-SHOT LEARNING - 10% TRAINING DATA

Across Weather and the ETT benchmarks, **PTEF** achieves the best overall average rank, with 2.450 for both MSE and MAE, as shown in Table 18. This indicates consistent performance under the 10% training-data. The strongest results are observed on high-frequency datasets. On **ETTm1**, PTEF obtains the lowest MSE and MAE at all prediction horizons. On **ETTm2**, it remains among the best methods, especially at horizons 96 and 192, although Time-LLM, TEMPO, and LLM4TS perform better in some longer-horizon cases. On **Weather**, PTEF gives the best results from horizons 96 to 336, while TEST performs better at horizon 720. On the hour-level datasets, **ETTh1** and **ETTh2**,

Table 16: Comparison of different decimal precisions. Model performance increases with decimal precision. Results using the full embedding model.

Decimals		10^2		10^5		10^7	
Dataset	T_{out}	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	**0.156**	↓**0.206**	**0.156**	**0.206**	↓**0.156**	↓**0.206**
	192	**0.199**	**0.244**	**0.199**	**0.244**	↓**0.199**	↓**0.244**
	336	0.251	0.283	**0.250**	**0.283**	↓**0.249**	↓**0.282**
	720	0.326	↓**0.336**	**0.325**	**0.336**	↓**0.325**	↓**0.336**
ETTh1	96	↓**0.419**	**0.427**	↓**0.419**	↓**0.427**	**0.419**	**0.427**
	192	**0.504**	**0.482**	**0.512**	**0.487**	↓**0.504**	↓**0.482**
	336	↓**0.531**	↓**0.498**	**0.539**	**0.506**	↓**0.531**	↓**0.498**
	720	**0.701**	**0.591**	↓**0.635**	↓**0.563**	**0.701**	**0.591**
ETTh2	96	**0.316**	↓**0.365**	↓**0.315**	**0.365**	**0.316**	↓**0.365**
	192	**0.395**	**0.406**	**0.397**	0.408	↓**0.394**	↓**0.406**
	336	0.407	**0.427**	↓**0.404**	**0.431**	**0.406**	↓**0.426**
	720	0.458	0.470	↓**0.458**	↓**0.469**	**0.458**	**0.470**
ETTm1	96	0.345	0.375	↓**0.343**	↓**0.374**	**0.345**	**0.375**
	192	**0.369**	**0.390**	**0.372**	0.391	↓**0.369**	↓**0.390**
	336	0.406	0.410	↓**0.405**	↓**0.410**	**0.406**	**0.410**
	720	↓**0.459**	↓**0.444**	**0.462**	**0.445**	↓**0.459**	↓**0.444**
ETTm2	96	↓**0.177**	**0.259**	**0.178**	**0.260**	↓**0.177**	↓**0.259**
	192	**0.235**	↓**0.300**	↓**0.234**	**0.301**	**0.235**	↓**0.300**
	336	↓**0.287**	↓**0.336**	↓**0.288**	**0.337**	**0.288**	↓**0.336**
	720	**0.404**	**0.400**	↓**0.399**	↓**0.398**	**0.404**	**0.400**

Table 17: Comparison of timestamp settings on 10% training data from ETTh1. Results using the fill embedding model.

Models		1st timestamp		Without timestamp		All timestamp	
Dataset	T_{out}	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	96	0.419	0.427	0.419	0.428	0.419	0.427
	192	0.504	0.482	0.505	0.482	0.504	0.482
	336	0.531	0.498	0.531	0.498	0.531	0.498
	720	0.701	0.591	0.701	0.591	0.701	0.591

PTEF remains competitive but is sometimes surpassed by TEMPO and Time-LLM, mainly at short horizons. However, PTEF still shows robustness at longer horizons, such as ETTh1-720, where it improves over LLM4TS. Overall, PTEF provides a strong accuracy–model size trade-off and outperforms classical models such as DLinear (Zeng et al., 2023) and PatchTST (Nie et al., 2023).

J.2 Full-shot Learning

In the full-shot setting, **PTEF** shows its strongest performance on the minute-level datasets. On **ETTm2**, it achieves strong short- and mid-horizon results, with 0.164/0.254 at $T_{out} = 96$ and the best MAE at $T_{out} = 192$ (0.291), as reported in Table 19. PTEF also remains competitive on **ETTm1**, where it obtains the best MAE at $T_{out} = 336$ (0.383) and stable performance at $T_{out} = 720$ (0.421/0.416). Compared with the few-shot setting, PTEF's advantage is smaller in the full-shot scenario. This may be due to its compact model size, which can limit its ability to benefit from larger training data compared with larger LLM-based models such as Time-LLM and LLM4TS. Increasing the number of CNN hidden channels improves full-shot performance Table 5. Despite this capacity limitation, PTEF achieves strong overall ranks (3.600 MSE / 3.000 MAE), outperforms classical models such as DLinear (Zeng et al., 2023) and One Fits ALL (Zhou et al., 2023), and remains competitive with PatchTST (Nie et al., 2023). These results show that PTEF provides a good accuracy–model size trade-off.

Table 18: **Few-shot Long-term forecasting using 10% of the training:** Results on Weather and ETT benchmarks (ETTh1, ETTh2, ETTm1, ETTm2) under prediction horizons $T_{out} \in \{96, 192, 336, 720\}$. We report MSE and MAE (↓, lower is better) and include **Avg. Rank** computed across all datasets and horizons. The best results in ↓red second-best in blue. Results for competing baselines are taken from LLM4TS (Chang et al., 2025).

Dataset	T_{out}	**PTEF (ours)**		LLM4TS		GPT4TS		DLinear		PatchTST		Time-LLM		TEMPO*		TEST	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	↓**0.156**	↓**0.206**	**0.158**	**0.207**	0.163	0.215	0.171	0.224	0.165	0.215	0.161	0.210	0.211	0.254	0.163	0.213
	192	↓**0.199**	↓**0.244**	**0.204**	0.249	0.210	0.254	0.215	0.263	0.210	0.257	**0.204**	**0.248**	0.254	0.298	0.230	0.263
	336	↓**0.250**	**0.283**	**0.254**	0.288	0.256	0.292	0.258	0.299	0.259	0.297	0.261	0.302	0.292	0.332	0.258	↓**0.282**
	720	0.325	0.336	0.322	0.336	0.321	0.339	0.320	0.346	0.332	0.346	**0.309**	**0.332**	0.370	0.379	↓**0.301**	↓**0.328**
ETTh1	96	0.419	**0.427**	**0.417**	0.432	0.458	0.456	0.492	0.495	0.516	0.485	0.448	0.460	↓**0.400**	↓**0.406**	0.455	0.457
	192	0.512	0.487	**0.469**	**0.468**	0.570	0.516	0.565	0.538	0.598	0.524	0.484	0.483	↓**0.426**	↓**0.421**	0.572	0.519
	336	0.539	0.506	**0.505**	**0.499**	0.608	0.535	0.721	0.622	0.657	0.550	0.589	0.540	↓**0.441**	↓**0.430**	0.578	0.531
	720	**0.635**	**0.563**	0.708	0.572	0.725	0.591	0.986	0.743	0.762	0.610	0.700	0.604	↓**0.443**	↓**0.451**	0.723	0.594
ETTh2	96	0.315	0.365	**0.282**	**0.351**	0.331	0.374	0.357	0.411	0.353	0.389	↓**0.275**	↓**0.326**	0.301	0.353	0.332	0.374
	192	0.397	0.408	**0.364**	0.400	0.402	0.411	0.569	0.519	0.403	0.414	0.374	↓**0.373**	↓**0.355**	**0.389**	0.401	0.433
	336	0.404	0.431	↓**0.374**	**0.416**	0.406	0.433	0.671	0.572	0.426	0.441	0.406	0.429	**0.379**	↓**0.408**	0.408	0.440
	720	0.458	0.469	0.445	0.461	0.449	0.464	0.824	0.648	0.477	0.480	**0.427**	**0.449**	↓**0.409**	↓**0.440**	0.459	0.480
ETTm1	96	↓**0.343**	↓**0.374**	0.360	**0.388**	0.390	0.404	0.352	0.392	0.410	0.419	**0.346**	**0.388**	0.438	0.424	0.392	0.401
	192	↓**0.372**	↓**0.391**	0.386	**0.401**	0.429	0.423	0.382	0.412	0.437	0.434	**0.373**	0.416	0.461	0.432	0.423	0.426
	336	↓**0.405**	↓**0.410**	0.415	**0.417**	0.469	0.439	0.419	0.434	0.476	0.454	**0.413**	0.426	0.515	0.467	0.471	0.444
	720	↓**0.462**	↓**0.445**	**0.470**	↓**0.445**	0.569	0.498	0.490	0.477	0.681	0.556	0.485	**0.476**	0.591	0.509	0.552	0.501
ETTm2	96	**0.178**	↓**0.260**	0.184	0.265	0.188	0.269	0.213	0.303	0.191	0.274	↓**0.177**	**0.261**	0.185	0.267	0.233	0.262
	192	↓**0.234**	↓**0.301**	**0.240**	↓**0.301**	0.251	0.309	0.278	0.345	0.252	0.317	0.241	0.314	0.243	0.304	0.303	**0.302**
	336	**0.288**	**0.337**	0.294	**0.337**	0.307	0.346	0.338	0.385	0.306	0.353	↓**0.274**	↓**0.327**	0.309	0.345	0.359	0.341
	720	**0.399**	0.398	↓**0.386**	**0.393**	0.426	0.417	0.436	0.440	0.433	0.427	0.417	↓**0.390**	↓**0.386**	0.395	0.452	0.419
Avg. Rank	–	↓**2.450**	↓**2.450**	**2.550**	↓**2.450**	5.000	4.900	6.100	6.800	6.600	6.650	2.850	**3.200**	4.450	4.500	5.700	4.750

Table 19: **Full-shot Long-term forecasting:** Results on Weather and ETT benchmarks (ETTh1, ETTh2, ETTm1, ETTm2) under prediction horizons $T_{out} \in \{96, 192, 336, 720\}$. We report MSE and MAE (↓, lower is better) and include **Avg. Rank** computed across all datasets and horizons. The best results in ↓red second-best in blue. Results for competing baselines are taken from LLM4TS (Chang et al., 2025).

Dataset	T_{out}	**PTEF (ours)**		LLM4TS		GPT4TS		DLinear		PatchTST		Time-LLM		TEMPO*		TEST	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	0.156	0.207	↓**0.147**	↓**0.196**	0.162	0.212	0.176	0.237	**0.149**	**0.198**	↓**0.147**	0.201	0.211	0.254	0.150	0.202
	192	0.200	0.247	**0.191**	**0.238**	0.204	0.248	0.220	0.282	0.194	0.241	↓**0.189**	↓**0.234**	0.254	0.298	0.198	0.246
	336	0.251	0.285	↓**0.241**	↓**0.277**	0.254	0.286	0.265	0.319	**0.245**	0.282	0.262	**0.279**	0.292	0.332	**0.245**	0.286
	720	0.324	0.336	**0.313**	**0.329**	0.326	0.337	0.333	0.362	0.314	0.334	↓**0.304**	↓**0.316**	0.370	0.379	0.324	0.342
ETTh1	96	**0.366**	↓**0.391**	0.371	0.394	0.376	0.397	0.375	0.399	0.370	0.399	↓**0.362**	**0.392**	0.400	0.406	0.372	0.400
	192	**0.401**	↓**0.412**	0.403	↓**0.412**	0.416	0.418	0.405	**0.416**	0.413	0.421	↓**0.398**	0.418	0.426	0.421	0.414	0.422
	336	0.426	0.429	↓**0.420**	↓**0.422**	0.442	0.433	0.439	0.443	**0.422**	0.436	0.430	**0.427**	0.441	0.430	**0.422**	0.437
	720	**0.435**	0.454	↓**0.422**	↓**0.444**	0.477	0.456	0.472	0.490	0.447	0.466	0.442	0.457	0.443	**0.451**	0.447	0.467
ETTh2	96	0.275	0.338	**0.269**	**0.332**	0.285	0.342	0.289	0.353	0.274	0.336	↓**0.268**	↓**0.328**	0.301	0.353	0.275	0.338
	192	0.332	**0.377**	↓**0.328**	**0.377**	0.354	0.389	0.383	0.418	0.339	0.379	**0.329**	↓**0.375**	0.355	0.389	0.340	0.379
	336	**0.352**	0.395	0.353	0.396	0.373	0.407	0.448	0.465	↓**0.329**	↓**0.380**	0.368	0.409	0.379	0.408	↓**0.329**	**0.381**
	720	0.390	0.428	0.383	0.425	0.406	0.441	0.605	0.551	**0.379**	**0.422**	↓**0.372**	↓**0.420**	0.409	0.440	0.381	0.423
ETTm1	96	0.295	**0.342**	**0.285**	0.343	0.292	0.346	0.299	0.343	0.290	**0.342**	↓**0.272**	↓**0.334**	0.438	0.424	0.293	0.346
	192	0.332	**0.363**	**0.324**	0.366	0.332	0.372	0.335	0.365	0.332	0.369	↓**0.310**	↓**0.358**	0.461	0.432	0.332	0.369
	336	0.365	↓**0.383**	**0.353**	0.385	0.366	0.394	0.369	0.386	0.366	0.392	↓**0.352**	**0.384**	0.515	0.467	0.368	0.392
	720	0.421	**0.416**	**0.408**	0.419	0.417	0.421	0.425	0.421	0.416	0.420	↓**0.383**	↓**0.411**	0.591	0.509	0.418	0.420
ETTm2	96	**0.164**	0.254	0.165	0.254	0.173	0.262	0.167	0.269	0.165	0.255	↓**0.161**	**0.253**	0.185	0.267	0.193	↓**0.237**
	192	↓**0.219**	**0.291**	**0.220**	0.292	0.229	0.301	0.224	0.303	**0.220**	0.292	↓**0.219**	0.293	0.243	0.304	0.257	↓**0.264**
	336	**0.271**	**0.326**	↓**0.268**	**0.326**	0.286	0.341	0.281	0.342	0.274	0.329	**0.271**	0.329	0.309	0.345	0.289	↓**0.295**
	720	0.367	0.382	↓**0.350**	0.380	0.378	0.401	0.397	0.421	0.362	0.385	**0.352**	**0.379**	0.386	0.395	0.375	↓**0.369**
Avg. Rank	–	3.600	3.000	**2.100**	**2.600**	5.850	5.700	6.650	6.500	3.150	3.900	↓**1.950**	↓**2.400**	7.450	6.900	4.700	4.250

K INFERENCE COST

MACs, or Multiply–Accumulate operations, represent the number of multiplication and addition operations performed by a model during inference. They are commonly used to estimate computational complexity and compare the efficiency of different architectures. In this work, MACs are calculated using the ETTh1 dataset with a prediction horizon of 720 time steps.

Table 21 compares the computational cost of the proposed main architecture with DLinear in terms of MACs. The results show that the main architecture requires significantly more computation, mainly due to the All-MiniLM-L6-v2 encoder, which accounts for almost all of the total cost.

Table 20: PTEF's MACs detailed for pruned embedding model and forecasting head.

Component	all-MiniLM-L6-v2	Forecasting Head	Total
MACs (G)	3.52	0.012	≈ 3.54

Table 21: MACs comparison calculated on ETTh1 with forecasting horizon 720. MACs are reported in giga MACs (G).

Model	PTEF	LLM4TS
MACs (G)	≈ 3.54	≈ 12.77