Decomposing the Time Series Forecasting Pipeline: A Modular Approach for Time Series Representation, Information Extraction, and Projection

Robert Leppich*, Michael Stenger*, André Bauer†, Daniel Grillmeyer* and Samuel Kounev*
*University of Würzburg, Würzburg, Germany
Email: {robert.leppich, michael.stenger, daniel.grillmeyer, samuel.kounev}@uni-wuerzburg.de
†Illinois Institute of Technology, Chicago, USA
Email: abauer7@iit.edu

***Abstract*—Time series forecasting demands effective sequence representation, meaningful information extraction, and precise future projection. This work decomposes the forecasting pipeline into three core stages: input sequence representation, information extraction and memory construction, and target projection. Within each stage, we investigate architectural configurations to assess the effectiveness of various modules, such as convolutional layers and self-attention mechanisms, across diverse forecasting tasks on seven benchmark datasets. Our models achieve state-of-the-art forecasting accuracy on the majority of benchmarks while maintaining a competitive parameter count and inference speed relative to strong baselines. The source code is publicly available at https://github.com/RobertLeppich/REP-Net.**

I. Introduction

Time series (TS) analysis has great potential in both science and industry, as it allows us to gain deeper insights into systems such as sensor networks [1], financial [2], and biological systems, such as the human body [3]. It comprises various disciplines, including TS forecasting [4], classification [5], and imputation [6]. Among these tasks, TS forecasting plays a vital role in predicting future trends and supporting data-driven decision-making, thereby enabling improved resource allocation and risk mitigation across domains such as finance, healthcare, and supply chain management.

TS data are often high-dimensional and characterized by complex temporal dependencies and intricate, potentially interfering variations, which makes modeling such dynamics particularly challenging.

Despite significant methodological advances, such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), challenges like the curse of dimensionality and vanishing/exploding gradients persist, restricting information flow across long sequences [7].

The emergence of the Transformer architecture [8], initially in Natural Language Processing (NLP) and later adopted in TS analysis, helped address the modeling of temporal variation. However, due to their pointwise attention mechanism, standard transformer-based models have difficulty fully capturing essential TS features and exhibit scalability issues with long sequences.

In response, research has progressed along several directions. **Attention-based approaches** introduce specialized mechanisms such as sparse or frequency-based attention and input segmentation through patching [9]–[11]. Simultaneously, **MLP-based models** and **frequency-domain methods** have demonstrated that simpler designs can achieve competitive results at significantly lower complexity [12]–[14]. Other recent developments include modern **convolutional approaches**, state space models, and large-scale **foundation models** [15]–[17].

However, these approaches typically optimize a single, fixed architecture whose effectiveness is highly task-dependent: each dataset poses its own challenges in terms of complexity and periodicity, and the forecasting horizon defines the temporal scale of patterns the model must learn, ranging from fine-grained short-term patterns to long-term seasonalities. For these reasons, we consider each combination of dataset and forecast horizon as an individual TS forecasting task, and we argue that the diversity of these tasks calls for a systematic investigation of how individual architectural components contribute to performance. Rather than proposing yet another monolithic architecture, we decompose the TS forecasting pipeline into modular stages and analyze the impact of architectural variations on forecasting performance across diverse scenarios.

Building on this insight, we propose **REP-Net** (**R**epresentation–M**e**mory–**P**rojection Network), a modular architecture that divides the forecasting process into three key modules: Representation of the input sequence, a Memory module that enriches the representation, and Projection onto the target output space.

We summarize the key contributions of this work as follows:

- We propose REP-Net, a novel architecture that decomposes the time series forecasting process into three distinct modules: Representation, Memory, and Projection. Each module is designed to flexibly incorporate a wide range of established and novel techniques for time series forecasting.
- We conduct a comprehensive empirical evaluation on seven established benchmark datasets for time series forecasting, comparing REP-Net with state-of-the-art (SOTA) models. REP-Net achieves competitive or superior performance

on the majority of benchmarks, demonstrating robustness and efficiency across diverse data distributions.
- We provide a systematic ablation study analyzing the individual and combined effects of architectural design choices, including attention mechanisms, time embeddings, GLU layers, and multi-scale patching, across 28 forecasting tasks, revealing task-specific insights into optimal configurations.
- To foster reproducibility and further research, we publicly release the REP-Net codebase[1], enabling researchers to benchmark and extend our work.

The remainder of this paper is organized as follows: Section II reviews related work on deep learning for time series forecasting. Section III introduces the modular REP-Net architecture and its components. Section IV details the experimental setup and provides a comprehensive evaluation of the proposed framework, including a systematic ablation study. Finally, Section VII concludes the paper with a summary of our findings and potential directions for future work.

II. Related Work

Deep learning approaches for long-term TS forecasting have evolved along several complementary directions. **Attention-based architectures** adapt the Transformer [8] to TS data through mechanisms that mitigate its quadratic complexity and inject temporal inductive biases: Informer [9] uses sparse attention, Autoformer [18] and FEDformer [19] integrate decomposition and frequency-domain operations, while PatchTST [10], iTransformer [11], and PathFormer [20] reformulate the input through patching, variate-wise tokens, or multi-scale patch routing. **MLP- and frequency-based models** challenge the need for attention: DLinear [12] shows that a linear layer can rival Transformer variants, TSMixer [13] applies MLP mixing along time and feature axes, CycleNet [14] exploits explicit cyclic priors, and TimeMixer [21] combines multi-scale downsampling with MLP mixing. **Convolutional and state-space models** leverage locality and structured recurrence: ModernTCN [15] revisits large-kernel convolutions, TimeCNN [22] adapts convolutional extractors to TS, and TimeMachine [16] uses structured state-space models for linear-time long-range modeling. Our prior work, TSRM [23], introduced a multi-scale CNN representation that we extend here through patch-based time-informed embeddings. **Foundation and LLM-based approaches** such as Chronos [17] and TEMPO [24] apply large-scale pretraining or adapt language models to TS inputs at the cost of substantially higher parameter counts.

In contrast, REP-Net does not commit to a single representational paradigm but decomposes the forecasting pipeline into three sequential modules and systematically exposes their architectural choices, enabling a principled analysis of which components benefit which forecasting scenarios.

III. Methodology

In the following, we introduce REP-Net, a novel, highly flexible architecture tailored to a wide range of time series (TS) forecasting tasks. REP-Net decomposes the forecasting pipeline into three sequential stages: **R**epresentation of the input sequence, a M**e**mory module that enriches the representation, and **P**rojection onto the target output space. This modular design offers two key advantages: First, it enables systematic ablation and comparison of different architectural choices at each stage, providing insights into which components contribute most to forecasting performance. Second, it allows practitioners to adapt individual modules to specific forecasting scenarios without redesigning the entire architecture. We define TS forecasting as follows: Given multivariate time series instances $\boldsymbol{x}_1 \dots \boldsymbol{x}_T$, where $\boldsymbol{x}_i \in \mathbb{R}^F$, F is the number of input features, and T is the input sequence length, the objective is to forecast $\hat{\boldsymbol{y}}_1 \dots \hat{\boldsymbol{y}}_H$, where $\hat{\boldsymbol{y}}_i \in \mathbb{R}^F$ and H is the forecasting horizon.

A. Architecture

As illustrated in Figure 1, we strictly divide the forecasting process into three distinct phases: (i) Representation (gray), (ii) Memory (red), and (iii) Projection (green). Unlike existing approaches that either rely on a single representation strategy [10] or fixed architectural designs [13], REP-Net provides a unified framework that allows systematic investigation of different design choices at each stage, enabling both empirical analysis of architectural components and practical adaptation to diverse forecasting scenarios. In the following, we describe the three modules in detail.

B. Representation

The Representation stage transforms the raw input sequence into a compact encoding that preserves structural and temporal dependencies essential for accurate forecasting. Given the inherently high dimensionality of time series data, this information must be compressed into a lower-dimensional representation suitable for subsequent stages while retaining essential patterns that the model can leverage during forecasting.

Prior work has explored various methods for effectively representing input sequences. Recurrent-based architectures, as proposed by [25], leverage the temporal modeling capabilities of recurrent units, such as GRUs and LSTMs, to capture sequential dependencies. To enhance pattern recognition, some approaches, as introduced by [22], adopt convolutional neural network (CNN)–based representations, while others combine recurrent and convolutional techniques to exploit both temporal and spatial features. In our prior work [23], we proposed a CNN-based representation layer that independently learns features at multiple levels of abstraction from an input sequence. This layer consists of several CNN modules with varying configurations, including small-kernel CNNs to capture fine-grained patterns and larger-kernel CNNs with high dilation rates to model higher-level structures, such as trends. The outputs of all CNN modules are concatenated, enabling subsequent stages to access information from all abstraction levels simultaneously. Other approaches employ functions, such as fast Fourier transformation, to determine different abstraction levels of the time series [20], [26]. Another approach involves dividing the time series into smaller segments, called patches. Each

[1]REP-Net GitHub: https://github.com/RobertLeppich/REP-Net.

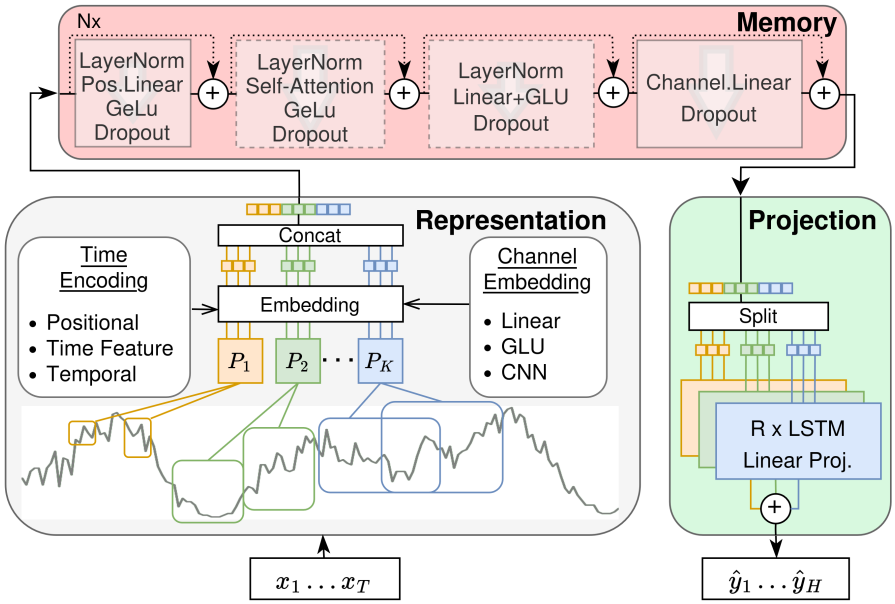

Fig. 1. Illustration of the proposed REP-Net Architecture, consisting of three modules: Representation (gray), Memory (red), and Projection (green).

patch is represented as a vector, which reduces complexity and enables the model to process meaningful chunks of the time series rather than individual time steps [10], [11]. However, existing patch-based methods typically rely on a single, fixed patch configuration, limiting their ability to capture patterns at multiple temporal scales simultaneously.

In this work, we investigate various patch-based techniques for representing time series data. Building on the multi-scale CNN representation from our prior work [23], we adopt patch extractors at multiple scales, but operate on patches rather than raw CNN features and augment them with temporal embeddings. Our method enables the extraction of patches from the input sequence $\boldsymbol{x} \in \mathbb{R}^{T \times F}$ at multiple levels of abstraction and sampling density. Specifically, we employ K distinct patch extractors P_k, each configured with a unique combination of patch length l_k, dilation rate d_k, and stride s_k. Given an input sequence of length T, patch extractor P_k produces $N_k = \lceil (T - d_k \cdot (l_k - 1))/s_k \rceil$ patches. Each patch $\boldsymbol{p}_{k,i} \in \mathbb{R}^{l_k \times F}$ represents a local window of the time series. In our experiments, we evaluate $K \in \{1, \dots, 5\}$.

The sub-sequences extracted by each patch extractor P_k are subsequently embedded using multiple embedding strategies E to evaluate their effectiveness across various forecasting horizons and datasets. Specifically, we consider three embedding strategies: E_1, a baseline method that applies linear layers, as employed in PatchTST [10]; E_2, an extension of E_1 that incorporates gated linear units (GLUs) to enhance feature extraction; and E_3, which utilizes CNNs for embedding, a technique we have shown to be effective in prior work [23]. For each strategy, the resulting series embedding is denoted as $\boldsymbol{E}_{x,k} \in \mathbb{R}^{N_k \times D}$, where D is the embedding dimension.

A key limitation of standard patches is that they lack explicit temporal context, as they represent patterns without knowing when these patterns occur. This is particularly important for datasets with strong periodicity (e.g., electricity consumption or traffic patterns), where the same numerical pattern may have different implications depending on the time of occurrence. To address this, the patch extractors also operate on corresponding temporal information $\boldsymbol{t} \in \mathbb{R}^{T \times F_{temp}}$, capturing time features such as day of the week and hour of the day using different time encodings. These time features are embedded using the same embedding method as the time series values, but with separate gradient flows during training, resulting in temporal embeddings $\boldsymbol{E}_{t,k} \in \mathbb{R}^{N_k \times D}$. The resulting time embeddings are concatenated along the feature dimension with the time series embeddings to form time-informed patches $\boldsymbol{E}_k = [\boldsymbol{E}_{x,k}; \boldsymbol{E}_{t,k}] \in \mathbb{R}^{N_k \times 2D}$. These patches jointly represent both an abstracted view of the input sequence and a corresponding abstraction of the temporal context.

The resulting patches from all K extractors are concatenated along the patch dimension into a single matrix $\boldsymbol{E}_{total} \in \mathbb{R}^{(\sum_k N_k) \times 2D}$, enabling the subsequent Memory modules to access embeddings from all levels of abstraction of both the TS and temporal information simultaneously.

C. Memory

After converting the input sequence into concatenated time-informed patches, the Memory module identifies and consolidates key patterns across the patch representations, producing a refined, information-rich encoding. This enrichment stage is critical, as it enables the final module to generate high-quality forecasts based on the most relevant temporal dependencies.

Previous works have employed Transformer-based attention [8], [10], [18], [19] and MLP-mixing approaches [13], [27] for this stage, typically on fixed input representations.

Our Memory module builds upon the time/feature MLP-mixing framework introduced by [13], extending it with multi-scale patch representations. We further enhance this design by incorporating normalization, activation, and dropout layers. The complete module architecture is depicted in Figure 1 (top). The module comprises four sequential layer blocks: (1) a positional linear transformation that establishes spatial relationships between patches; (2) an optional sparse self-attention block, inspired by [28], that captures temporal dependencies across patches; (3) an optional feature-level linear layer followed by a gated linear unit (GLU) that facilitates selective information retention by allowing the model to discard irrelevant features; and (4) a final feature-level linear transformation that supports two modes: (i) operating across all features jointly, enabling the capture of inter-feature dependencies, or (ii) processing each feature embedding independently for channel-independent modeling. All blocks and the entire module are equipped with residual connections to facilitate information and gradient flow. Each block's contribution can be isolated through ablation, enabling systematic analysis of their individual and combined effects. The module is stacked N times, enabling deep architectures to capture hierarchical information and patterns.

In our experiments, we assess the impact of the attention and GLU blocks on model performance. Additionally, we vary the number of stacked memory modules (N) to evaluate their effect across different datasets and forecasting horizons.

D. Projection

The final step involves generating predictions $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_H$ from the enriched representations. Recent work has shown that simple linear projections can achieve competitive performance [10], [12], [23], often matching Transformer-based autoregressive approaches [18], [19].

The projection module, shown in Figure 1 (green), comprises K separate projection blocks, where K corresponds to the number of patch extractors in the Representation module. Before entering the projection module, the concatenated patches are split back into their original partitions as produced by the patch extractors. An individual projection block then processes each of the resulting K patch sequences. The outputs of these K projection blocks are summed element-wise to produce the final forecast $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_H$. This summation encourages each projection block to produce complementary predictions that, when combined, capture both fine-grained and coarse-grained temporal patterns.

We investigate various configurations of the projection block, each comprising R LSTM layers followed by a linear layer. The LSTM layers are intended to capture temporal dependencies within each patch sequence, thereby enhancing the forecasting capability of the subsequent linear layer. We evaluate the effect of varying the number of LSTM layers with $R \in \{0, 1, 2, 3\}$, where $R = 0$ indicates the absence of LSTM layers. Our experiments assess the impact of these configurations on forecasting performance across different benchmark datasets and forecast horizons. In the following section, we evaluate the proposed architecture and its modular configurations through a series of experiments on diverse time series forecasting benchmarks.

IV. Experiments

Following the modular design of REP-Net introduced in the previous section, we now evaluate its performance across various forecasting tasks. In order to assess the effectiveness of our proposed architecture, we conducted a series of experiments using publicly accessible and well-established benchmark datasets from different fields: ECL [29], ETT [9] (four subsets: ETTm1, ETTm2, ETTh1, ETTh2), Weather [18], and Traffic [30]. All datasets were sourced from [18].

A. Experimental Setup

For all experiments, we applied early stopping with a patience of three epochs, using the validation loss as the evaluation metric. The training was terminated when the relative improvement of the loss was less than 1%. Hyperparameter tuning was performed via random search over the configuration space summarized in Table I.

TABLE I
Reproducibility Box: Summary of the hyperparameter search space and experimental settings.

Category	Hyperparameters / Settings
General	Dropout $\in \{0.25, 0.33, 0.5, 0.66, 0.9\}$ Optimizer: Adam, Scheduler: ReduceLROnPlateau Patience: 3 epochs, Min. Improvement: 1%
Representation	Patch Extractors $K \in \{1, 2, 3, 4, 5\}$ Patch Sizes $\in \{3, 5, 10, 15, 20, 48, 64\}$ Embedding Size $D \in \{4, 8, 16, 32, 64, 128\}$ Time Embeddings $\in \{timeF, tempEmb, posEmb\}$
Memory	Stacked Modules $N \in \{0, 1, 2, 3, 4, 5, 8, 10\}$ Attention Heads $\in \{4, 8, 16, 32\}$ Feature Mixing $\in$ {Jointly across all, Independent}
Projection	LSTM Layers $R \in \{0, 1, 2, 3\}$

For the representation module, we explored various patch extractors P_k and representation techniques as described in Section III-B. Additionally, we investigate different encoding sizes and time embedding techniques. For the Memory module, we vary the number of memory modules (denoted by N), the inclusion of the attention block, and the presence of the GLU block, as described in Section III-C. We also explore different numbers of attention heads in the self-attention layer. Additionally, we examine the impact of configuring the feature-level linear layer in the memory module to either operate jointly across all features or to operate independently for each feature. For the projection module, we vary the number of LSTM layers.

Learning rates were dynamically adapted during training using the learning rate scheduler *ReduceLROnPlateau* from *PyTorch* with a patience of one epoch. All models were trained with the Adam optimizer [31] using the Huber loss, which provides robustness against outliers while maintaining smooth gradients. Hyperparameter search was conducted on a single GPU with early stopping as described above; full configuration logs are provided in the released codebase.

B. Long-Term Time Series Forecasting

To evaluate the performance of our architecture for long-term TS forecasting, we adopted the procedure by [11]: To support a fair comparison with other approaches, we maintain the input length for all approaches at $T = 96$, while varying the prediction horizon $H \in \{96, 192, 336, 720\}$. For the evaluation, we consider seven datasets (i.e., ECL, ETTm1, ETTm2, ETTh1, ETTh2, Weather, and Traffic) and compare against multiple SOTA forecasting techniques: TEMPO [24], CycleNet [14], TimeMixer [21], TimeMachine [16], TimeCNN [22], TSRM [23], PathFormer [20], iTransformer [11], and PatchTST [10].

C. Results

Table II reports the overall best results across all module configurations compared to SOTA architectures. REP-Net consistently delivers strong performance across most datasets and prediction lengths, demonstrating superior forecasting accuracy, particularly at the longest horizon of 720. For shorter horizons, our method largely matches SOTA performance. The small performance differences at these horizons may indicate that the forecasting task becomes easier when temporal dependencies are more immediate, reducing the advantage of sophisticated architectural designs.

Table III presents results for long-term forecasting with variable input length and prediction horizon $T, H \in \{96, 192, 336, 720\}$. REP-Net leads on most datasets under the variable-lookback protocol, with the only notable gap on ETTh2, where TimeMachine and PathFormer show stronger performance in Table II. This dataset exhibits high volatility and non-stationary patterns that may benefit from the specific inductive biases of these architectures.

Comparing Tables II and III quantifies the benefit of longer lookback windows: averaged over all horizons, extending from fixed $T = 96$ to variable lookback reduces REP-Net's MSE by 7–8% on ECL, Weather, ETTm1, and ETTm2, 4.3% on ETTh2, and only 2.6% on ETTh1. The larger gains on datasets with pronounced periodicity (ECL, Weather, ETTm1/2) indicate that the multi-scale patch extractors effectively exploit the additional temporal context, whereas ETTh1, whose predictive information is concentrated in recent samples, benefits comparatively little.

Beyond performance, we evaluated model complexity with $H = T = 96$. As summarized in Table IV, REP-Net achieves a strong balance between parameter count, inference speed, and memory footprint.

Although REP-Net has more parameters than the purely MLP-based TimeMixer and CycleNet, its per-iteration time is competitive with CycleNet and roughly 40% lower than TimeMixer, while its memory footprint remains below 50 MB. We attribute this to two properties: the K patch extractors and projection blocks execute as shallow parallel branches rather than deep sequential stacks, keeping the critical-path depth short, and the projection module operates directly on the patch-level representation instead of autoregressively over the forecast horizon. The gap to PathFormer is particularly striking ($\sim 4\times$ fewer parameters, an order of magnitude less memory, $>13\times$ faster inference), while TSRM carries roughly twice the parameters of REP-Net at comparable runtime.

V. Model Analysis

Having established that REP-Net achieves competitive overall performance, we now examine which architectural components drive these results and how their effectiveness varies across tasks. To assess the impact of individual architectural modifications, their interdependencies, and their influence on performance, Table V reports the relative MSE change achieved by each configuration relative to a baseline. For each architectural choice (column), the baseline is the best-performing configuration *without* that component; positive values indicate an *MSE reduction* (i.e., an improvement) when adding the component, while negative values indicate a degradation. Non-negative values are shown in green, from light for a small MSE reduction relative to the baseline to dark for larger reductions. Negative values shown in yellow indicate small performance degradations, while values shown in orange and red denote larger deteriorations. For example, in the first row of the table, for the ECL dataset with a prediction horizon of $H = 96$, the *Time Embedding* column shows a value of +10.4% in dark green. This indicates that the best configuration *with* Time Embedding achieves a 10.4% lower MSE than the best configuration *without* Time Embedding.

As previously discussed, we consider each combination of dataset and forecasting horizon as a separate task, reflecting their distinct challenges. Consequently, we perform a task-specific analysis to assess the impact of each architectural change on each task. In the following, we provide a detailed discussion of each architectural variation.

Is attention all we need? The attention block within the Memory module enables dependency modeling and contextual learning across diverse representations and has demonstrated effectiveness in prior TS forecasting approaches [10], [11], [19]. However, self-attention incurs substantial computational overhead, which must be justified by corresponding performance gains. The percentage changes relative to the best configuration, shown in the *Attention* column of Table V, indicate a general decrease in performance across most tasks when attention mechanisms were incorporated into the architecture. The degradation is particularly pronounced in the ECL tasks, where performance dropped by up to 9.4%. For most tasks, the reduction was minor; however, in certain cases, such as the Traffic dataset with longer forecast horizons, it exhibited a performance improvement of up to 3.2%. We attribute this to the already strong inductive bias introduced by multi-scale patches, which capture much of the temporal context that self-attention would otherwise provide, and to the short input length $T = 96$, which limits the benefit of global attention compared to more structured aggregation. Furthermore, combining CNN embeddings with attention mechanisms adversely affected performance. Across the 28 tasks, we measured a Pearson correlation of -0.54 between the per-task MSE gains of attention and CNN embeddings, indicating that their functions

TABLE II

PERFORMANCE COMPARISON FOR THE MULTIVARIATE FORECASTING TASK WITH PREDICTION HORIZONS $H \in \{96, 192, 336, 720\}$ AND FIXED LOOKBACK WINDOW $T = 96$. RESULTS ARE AVERAGED OVER ALL PREDICTION HORIZONS. FOR EACH METRIC, WE BOLD THE BEST VALUE AND ANY RESULT WITHIN 1% OF IT (I.E., EFFECTIVELY TIED); THE BEST NON-BOLD RESULT IS UNDERLINED.

Dataset	REP-Net		TEMPO (ICLR 2024)		CycleNet (NeurIPS 2024)		TimeMixer (ICLR 2024)		TimeMachine (ECAI 2024)		TimeCNN		TSRM		PathFormer		iTransformer		PatchTST	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ECL	**0.165**	0.266	0.216	0.308	0.167	**0.258**	0.182	0.272	0.170	0.263	0.170	0.264	0.175	0.270	0.182	0.269	0.178	0.270	0.190	0.275
Weather	**0.234**	0.265	0.287	0.318	0.243	0.271	0.240	0.272	0.243	0.273	0.252	0.278	0.239	0.267	0.239	**0.262**	0.258	0.278	0.260	0.285
Traffic	**0.372**	0.280	0.503	0.357	0.472	0.301	0.485	0.298	0.428	0.281	0.410	**0.273**	0.545	0.324	0.501	0.299	0.428	0.282	0.466	0.293
ETTh1	**0.416**	0.425	0.446	0.455	0.432	0.428	0.446	0.440	**0.416**	**0.419**	0.441	0.436	0.435	0.430	0.439	0.430	0.454	0.448	0.454	0.444
ETTh2	0.352	0.389	0.361	0.398	0.382	0.406	0.365	0.395	**0.344**	**0.379**	0.376	0.402	0.371	0.396	**0.344**	**0.379**	0.383	0.406	0.378	0.402
ETTm1	**0.366**	0.390	0.373	0.396	0.379	0.396	0.381	0.396	0.374	0.392	0.383	0.392	0.380	0.392	0.382	**0.386**	0.407	0.410	0.384	0.396
ETTm2	0.264	**0.314**	0.280	0.328	0.266	**0.314**	0.275	0.323	0.268	0.318	0.280	0.325	**0.245**	**0.314**	0.273	**0.314**	0.290	0.332	0.284	0.327

TABLE III

PERFORMANCE COMPARISON FOR THE MULTIVARIATE FORECASTING TASK WITH PREDICTION HORIZONS $H \in \{96, 192, 336, 720\}$ AND VARIABLE LOOKBACK WINDOW $T \in \{96, 192, 336, 720\}$. RESULTS ARE AVERAGED OVER ALL PREDICTION HORIZONS. FOR EACH METRIC, WE BOLD THE BEST VALUE AND ANY RESULT WITHIN 1% OF IT (I.E., EFFECTIVELY TIED); THE BEST NON-BOLD RESULT IS UNDERLINED.

Dataset	REP-Net		TimeMixer (ICLR 2024)		CycleNet (NeurIPS 2024)		PatchTST (ICLR 2023)	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ECL	**0.152**	0.251	0.156	**0.247**	0.157	0.250	0.159	0.253
Weather	**0.217**	**0.252**	0.222	0.262	0.223	0.264	0.226	0.264
ETTh1	**0.405**	**0.424**	0.411	**0.423**	0.415	0.426	0.413	0.434
ETTh2	0.337	0.388	**0.316**	**0.384**	0.355	0.398	0.330	**0.381**
ETTm1	**0.338**	**0.371**	0.348	0.376	0.354	0.380	0.351	0.381
ETTm2	**0.243**	**0.305**	0.256	0.316	0.251	0.309	0.307	0.315

TABLE IV

MODEL COMPLEXITY COMPARISON AT $H = T = 96$: PARAMETER COUNT, PER-ITERATION INFERENCE TIME, AND PEAK MEMORY USAGE.

Model	Params	Time/iter	Memory
REP-Net	347K	0.009 s	44 MB
TimeMixer	90K	0.015 s	63 MB
CycleNet	113K	0.009 s	25 MB
TSRM	752K	0.012 s	51 MB
PathFormer	1437K	0.124 s	358 MB

overlap and that a single approach is more effective for contextual learning.

Are time-informed patches superior to non-informed patches? Time-informed patches incorporate a time embedding to provide the model with temporal context. We analyze whether including time embeddings yields performance gains over standard patches without temporal information. As shown in Table V, nearly all tasks benefit from time-informed patches compared to standard patches. The Traffic dataset, in particular, demonstrates considerable performance improvements. Time-feature and temporal embeddings consistently yield the best results across most tasks, whereas positional embeddings alone provide smaller gains.

Does the recurrence mechanism of LSTMs enhance performance in patch-based time series forecasting? LSTMs facilitate the modeling of temporal context. In our approach, we incorporate LSTM layers within the projection module to capture temporal dependencies between patches before applying the final linear projection. As shown in Table V, the ECL and ETTh1 datasets benefit from the inclusion of LSTM layers, while most of the remaining datasets perform better without LSTMs.

Is it sometimes better to forget? By integrating GLU layers into specific representation embeddings and the memory module, the model effectively filters out irrelevant information, thereby enhancing information flow. As shown in Table V, incorporating GLU within the Memory module yields a notable performance improvement across almost all tasks. In particular, the ECL and Traffic datasets exhibit substantial gains.

How many representatives do we need? Most existing work in patch-based time series forecasting uses a fixed patch length to represent the input sequence [10], [11]. In our work, we use multiple patch extractors in order to receive representations on different abstraction levels, capturing fine-grained features with smaller patches and features like trend with larger patches. The results in Table V (Multi Patch) show superior performance of our architecture when using more than one patch extractor. For most tasks, three or four patch extractors yielded the best-performing setups.

Do we need the memory stack? Stacking memory modules increases model depth, enabling it to capture more complex data dependencies. In Table V, we compare the performance of a single memory module (*Memory*) against no memory module, as well as multiple stacked memory modules (*Multi Memory*) against a single memory module. The results demonstrate a clear advantage of using a single memory module over no memory module. However, the comparison between a single and multiple memory modules shows task-dependent performance, with no consistent superiority across tasks.

Do CNN-based embeddings offer any advantages? CNN-based embeddings leverage CNNs' feature extraction capabilities to enhance representation learning. In this work, we examine three distinct CNN-based embedding strategies. The ablation results presented in Table V reveal task-dependent performance: certain datasets, such as ECL, ETTh1, and ETTm2, benefit considerably from CNN-based embeddings, while others, such as Traffic, exhibit performance degradation when these embeddings are employed. Notably, most tasks achieved their best performance using our CNN-based embedding architecture consisting of two convolutional layers, each followed by a max-pooling layer. In contrast, the CNN variant

TABLE V

COMPARISON OF ARCHITECTURAL VARIANTS ACROSS TASKS. THE REPORTED VALUES REPRESENT THE PERCENTAGE CHANGE IN PERFORMANCE RELATIVE TO A BASELINE MODEL WITHOUT THE CORRESPONDING CONFIGURATION. COLOR CODING IS USED TO HIGHLIGHT PERFORMANCE IMPACT: NON-NEGATIVE VALUES ARE SHOWN IN GREEN, FROM LIGHT FOR A SMALL IMPROVEMENT IN MSE RELATIVE TO THE BASELINE TO DARK FOR LARGER IMPROVEMENTS. NEGATIVE VALUES SHOWN IN YELLOW INDICATE SMALL PERFORMANCE DEGRADATIONS, WHILE VALUES SHOWN IN ORANGE AND RED DENOTE LARGER DETERIORATIONS.

Dataset		Attention	Time Embedding	GLU	LSTM	Multi Patch	Memory	Multi Memory	CNN Embedding
ECL	96	-1.4	+10.4	+0.5	+7.6	+0.3	-0.3	+0.3	+3.4
	192	+0.8	+1.6	+2.1	-0.8	-0.8	+1.5	-0.8	+0.8
	336	-5.7	+2.4	+13.5	+1.4	+4.0	+1.4	+4.3	-3.0
	720	-9.4	+4.6	+2.0	-2.0	+2.0	+7.5	+2.0	+4.6
ETTh1	96	-0.1	+0.1	+0.0	-0.2	-0.1	+0.1	+0.1	-2.7
	192	-1.3	+0.8	0.0	+0.8	+1.7	-1.3	+0.5	-1.7
	336	-0.6	+2.3	+0.8	+0.6	-0.6	+2.3	-0.6	+0.6
	720	-3.1	+3.6	+0.5	+3.1	+9.8	-1.5	+4.6	+6.3
ETTh2	96	-0.3	-0.3	-0.7	-1.1	+1.1	+0.4	+0.8	-0.6
	192	-0.3	+0.0	+0.3	-1.6	+0.3	+1.1	+0.0	-0.0
	336	+1.1	-1.6	+1.6	-1.7	+2.7	-0.3	+1.9	-3.3
	720	-0.3	-0.3	+0.3	-1.1	+0.3	+0.1	+0.3	-1.1
ETTm1	96	-0.2	-0.2	+0.2	-1.9	+1.4	+2.4	-1.9	-1.4
	192	-3.6	-0.0	-0.0	-0.8	+0.8	+0.8	-0.8	+0.1
	336	+0.2	+1.3	-0.2	-0.8	+1.7	-0.5	+0.3	+0.5
	720	-0.2	-0.2	+0.1	-0.2	+1.7	+0.1	+0.1	+0.1
ETTm2	96	-2.9	+4.4	+1.1	-4.0	+4.0	+3.6	-1.5	+1.5
	192	+1.0	+2.8	+1.7	-1.6	+1.9	+2.8	-2.0	+1.0
	336	-0.5	+2.2	+1.4	-1.4	+0.5	+2.0	-0.1	+0.5
	720	-2.7	+3.0	-0.1	-4.2	+3.5	+0.5	+3.7	+2.3
Traffic	96	+0.4	+13.7	+2.4	+0.5	+14.0	+3.2	-0.4	-2.0
	192	-1.4	+10.7	+2.7	+2.5	+1.4	-2.5	+1.0	-2.0
	336	+3.2	+9.1	+5.3	-2.4	+3.7	-1.3	+3.7	-2.4
	720	+2.1	+9.5	+4.4	-2.1	+4.5	-0.5	+2.6	-4.5
Weather	96	-0.5	+0.6	+0.6	-2.3	-0.1	+5.1	-2.3	-0.1
	192	-0.7	+0.7	+0.8	-1.8	+0.3	+2.8	-0.7	-0.7
	336	-1.2	+1.4	+0.0	-1.3	-0.5	-0.5	-0.9	+0.5
	720	-0.5	+1.7	-0.8	-1.9	+0.9	+2.5	-0.5	-0.5

employing two convolutional layers, each followed by a GELU activation, rarely yielded competitive results.

From ablations to configuration profiles. Consolidating the observations above, Table VI maps salient dataset characteristics to the modules that provide the largest gains. Datasets with strong periodicity (ECL, Traffic) benefit disproportionately from time-informed patches, with MSE reductions of up to 13.7% on Traffic and 10.4% on ECL, while datasets dominated by multiple concurrent temporal scales (ETTh1, ETTm2, Traffic) require two or more patch extractors. GLU layers are broadly helpful but show their largest effect on datasets with many potentially irrelevant feature channels, whereas highly volatile datasets such as ETTh2 do not benefit from additional memory depth or self-attention.

TABLE VI

CONFIGURATION PROFILES DERIVED FROM THE ABLATION IN TABLE V: DATASET CHARACTERISTICS AND THE REP-NET MODULES THAT PROVIDE THE LARGEST PERFORMANCE GAINS.

Data characteristic	**Example datasets**	**Beneficial modules**
Strong periodicity	ECL, Traffic	Time embeddings, GLU
Multi-scale dynamics	ETTh1, ETTm2, Traffic	Multi-patch ($K \geq 2$)
High volatility	ETTh2	Shallow memory, no attention
Many feature channels	ECL, Traffic	GLU, independent mixing

VI. LIMITATIONS

Although REP-Net achieves strong performance on the benchmark datasets used in our experiments, several potential limitations remain, as outlined below.

Generalizability and Dataset Diversity: Our evaluation focuses on established long-term forecasting benchmarks (ECL, ETT, Weather, Traffic), which primarily represent periodic sensor and energy grid data and, as noted by [32], may not fully capture the complexity of heterogeneous real-world scenarios. Future work will extend the evaluation to a broader suite of non-standard benchmarks, including multi-domain and short-term forecasting tasks, to further validate the robustness and adaptability of the modular approach.

Model analysis: Despite substantial efforts to analyze the impact of various architectural modifications on model performance, our analysis is restricted to a fixed set of architectural variations and does not isolate the side effects of different hyperparameters. Further experiments are necessary to investigate the influence of specific hyperparameters on model performance within particular architectural configurations. In addition, ablation results are reported for a single seed per configuration, so small differences ($< 1\%$) should be interpreted with caution as they fall within typical training variance. Likewise, the main comparison tables report single-seed results without confidence intervals, so small gaps ($< 1\%$) between methods should not be interpreted as definitive rankings.

Complexity and runtime: While the current system demonstrates promising resource efficiency and low inference latency, our comparison with other methods revealed instances of superior inference speed and reduced memory usage. These findings warrant further investigation to reduce REP-Net's computational complexity.

VII. Conclusion

We propose REP-Net, a modular architecture for time series forecasting that decomposes the pipeline into Representation, Memory, and Projection stages. REP-Net achieves competitive or superior performance compared to SOTA methods across established benchmarks while maintaining low computational complexity.

Our analysis reveals that GLU layers and time-informed patches consistently improve performance across most datasets, with time embeddings yielding substantial gains of up to 13.7% on datasets with strong periodicity (e.g., Traffic, ECL), and that multi-patch extractors benefit tasks requiring both fine-grained and trend-level pattern recognition. In contrast, self-attention mechanisms show limited benefit for most tasks, with CNN-based embeddings often providing comparable contextual learning at lower computational cost. These findings highlight the importance of adapting architectural configurations to the unique properties of both the dataset and forecasting horizon.

Future directions. Several promising avenues emerge from this work: (1) extending the modular framework to other time series tasks such as classification and imputation, (2) investigating learned patch extraction strategies that adapt to the input, and (3) developing automated configuration selection methods that recommend optimal architectural choices based on dataset characteristics.

References

[1] S. Papadimitriou and P. Yu, "Optimal multi-scale patterns in time series streams," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006, pp. 647–658.

[2] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 358–369.

[3] S. Ek, F. Portet, and P. Lalanda, "Transformer-based models to deal with heterogeneous environments in human activity recognition," *Personal and Ubiquitous Computing*, vol. 27, no. 6, pp. 2267–2280, 2023.

[4] X. Liu and W. Wang, "Deep time series forecasting models: A comprehensive survey," *Mathematics*, vol. 12, no. 10, p. 1504, 2024.

[5] N. Mohammadi Foumani, L. Miller, C. W. Tan, G. I. Webb, G. Forestier, and M. Salehi, "Deep learning for time series classification and extrinsic regression: A current survey," *ACM Computing Surveys*, vol. 56, no. 9, pp. 1–45, 2024.

[6] Y. Jaradat, M. Masoud, A. Manasrah, M. Alia, and I. Jannoud, "Review of data imputation techniques in time series data: Comparative analysis," *The Eurasia Proceedings of Science Technology Engineering and Mathematics*, vol. 27, pp. 122–129, 2024.

[7] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[9] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 11 106–11 115.

[10] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," *arXiv preprint arXiv:2211.14730*, 2022.

[11] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "itransformer: Inverted transformers are effective for time series forecasting," *arXiv preprint arXiv:2310.06625*, 2024.

[12] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 9, 2023, pp. 11 121–11 128.

[13] S.-A. Chen, C.-L. Li, S. O. Arik, N. C. Yoder, and T. Pfister, "TSMixer: An all-MLP architecture for time series forecast-ing," *Transactions on Machine Learning Research*, 2023. [Online]. Available: https://openreview.net/forum?id=wbpxTuXgm0

[14] S. Lin, W. Lin, X. Hu, W. Wu, R. Mo, and H. Zhong, "Cyclenet: Enhancing time series forecasting through modeling periodic patterns," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 106 315–106 345. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/bfe7998398779dde03cad7a73b1f81b6-Paper-Conference.pdf

[15] D. Luo and X. Wang, "Moderntcn: A modern pure convolution structure for general time series analysis," in *The twelfth international conference on learning representations*, 2024, pp. 1–43.

[16] M. A. Ahamed and Q. Cheng, "Timemachine: A time series is worth 4 mambas for long-term forecasting," in *27th European Conference on Artificial Intelligence*, vol. 392, 2024, pp. 1688–1695.

[17] A. F. Ansari, L. Stella, C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor *et al.*, "Chronos: Learning the language of time series," *arXiv preprint arXiv:2403.07815*, 2024.

[18] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 419–22 430, 2021.

[19] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *International Conference on Machine Learning*. PMLR, 2022, pp. 27 268–27 286.

[20] P. Chen, Y. Zhang, Y. Cheng, Y. Shu, Y. Wang, Q. Wen, B. Yang, and C. Guo, "Pathformer: Multi-scale transformers with adaptive pathways for time series forecasting," in *International Conference on Learning Representations*, 2024.

[21] S. Wang, H. Wu, X. Shi, T. Hu, H. Luo, L. Ma, J. Y. Zhang, and J. ZHOU, "Timemixer: Decomposable multiscale mixing for time series forecasting," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=7oLshfEIC2

[22] A. Hu, D. Wang, Y. Dai, S. Qi, L. Wen, J. Wang, Z. Chen, X. Zhou, Z. Xu, and J. Duan, "Timecnn: Refining cross-variable interaction on time point for time series forecasting," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2410.04853

[23] R. Leppich, M. Stenger, D. Grillmeyer, V. Borst, and S. Kounev, "Tsrm: A lightweight temporal feature encoding architecture for time series forecasting and imputation," *arXiv preprint arXiv:2504.18878*, 2025.

[24] D. Cao, F. Jia, S. O. Arik, T. Pfister, Y. Zheng, W. Ye, and Y. Liu, "Tempo: Prompt-based generative pre-trained transformer for time series forecasting," in *The Twelfth International Conference on Learning Representations*, 2024.

[25] S. Y. Jhin, S. Kim, and N. Park, "Addressing prediction delays in time series forecasting: A continuous gru approach with derivative regularization," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 1234–1245.

[26] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, "Timesnet: Temporal 2d-variation modeling for general time series analysis," *arXiv preprint arXiv:2210.02186*, 2022.

[27] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit *et al.*, "Mlp-mixer: An all-mlp architecture for vision," *Advances in neural information processing systems*, vol. 34, pp. 24 261–24 272, 2021.

[28] S. Wu, X. Xiao, Q. Ding, P. Zhao, Y. Wei, and J. Huang, "Adversarial sparse transformer for time series forecasting," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[29] D. Dua, C. Graff *et al.*, "Uci machine learning repository," 2017.

[30] "California department of transportation," http://pems.dot.ca.gov, [Online; accessed 1. Feb 2024].

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] C. Bergmeir, "Fundamental limitations of foundational forecasting models: The need for multimodality and rigorous evaluation," https://cbergmeir.com/talks/neurips2024/, 2024, [Online; accessed 01. Feb 2025].