REVISITING RECURRENT MODELS FOR LONG-TERM TIME SERIES FORECASTING THROUGH REPRESENTATION ACCESSIBILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

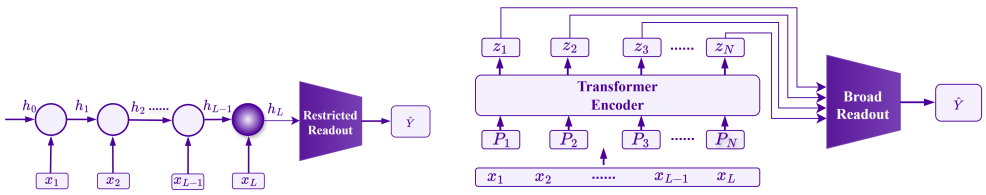
Transformer-based architectures have become prominent in long-term time series forecasting (LTSF), and their empirical advantages are often attributed to self-attention. This paper revisits that attribution by identifying a less visible architectural confounder in common LTSF comparisons: representation accessibility. In many direct multi-horizon recurrent baselines, the prediction module is conditioned on a compressed recurrent summary, typically the final encoder state, or on a decoder initialized from that state. In contrast, patch-based Transformer forecasters such as PatchTST expose representations from multiple temporal positions directly to a non-autoregressive prediction head. This difference changes not only the sequence modeling mechanism, but also which historical representations are accessible at prediction time. To isolate this factor, we introduce a Full-State Readout (FSR) framework built from standard recurrent encoders. FSR keeps key ingredients of strong LTSF forecasters fixed, including patching, channel independence, normalization, and a Flatten–Linear prediction head, while varying only the number of recurrent hidden states exposed to the head. Across six LTSF benchmarks, FSR with a GRU backbone matches or outperforms representative Transformer baselines. Expanding representation accessibility generally reduces forecasting errors for GRU, and full-state readout also improves vanilla RNN and LSTM encoders. These results show that the recurrent–Transformer gap in LTSF cannot be attributed solely to recurrence versus self-attention. More broadly, they suggest that final-state bottlenecks can systematically underestimate recurrent models, and that representation accessibility should be controlled when comparing sequence modeling architectures for forecasting.

1 INTRODUCTION

Long-Term Time Series Forecasting (LTSF) aims to predict multiple future time steps from a long history of observations, with applications in energy, transportation, weather, and industrial systems (Lim & Zohren, 2021). Transformer-based architectures have become a dominant line of work in LTSF, with studies exploring efficient attention, sequence decomposition, frequency-domain modeling, patching, and cross-variable interactions (Wen et al., 2023; Zhou et al., 2021; 2022; Nie et al., 2023). Their empirical success is often attributed to self-attention, which can directly model global interactions across temporal positions. In contrast, recurrent neural networks update hidden states sequentially and are often regarded as less suitable for long-horizon forecasting.

However, recent studies show that competitive LTSF performance does not necessarily require self-attention, as demonstrated by linear models and factorized temporal and channel mixing (Zeng et al., 2023; Li et al., 2025). Forecasting accuracy also depends on input organization, normalization, and non-stationarity handling (Nie et al., 2023; Kim et al., 2022; Liu et al., 2022b; Lin et al., 2026). These results raise a question: when Transformers outperform recurrent models in LTSF benchmarks, *how much of the gap should be attributed to self-attention itself, and how much to other architectural factors in the forecasting pipeline*?

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.



(a) Restricted Readout

(b) Broad Readout

Figure 1: Comparison of Restricted and Broad Readout Architectures.

In this work, we focus on one such factor: *Representation Accessibility*, defined as the range of encoder representations directly available to the forecasting module. Recurrence determines how hidden states are computed; it does not require prediction to rely exclusively on the final state. Nevertheless, many direct multi-horizon recurrent baselines expose only the final encoder state, imposing a restricted readout interface on an encoder that produces a sequence of hidden states. As illustrated in Figure 1, this differs from patch-based Transformer forecasters such as PatchTST, which expose multiple temporal representations to a non-autoregressive Flatten–Linear head. Such comparisons therefore vary both the encoder mechanism and the access that the predictor has to encoded history.

Table 1: Architectural comparison.

Method	Patch/ segment	Multi-state readout	Non-AR head
RNNsearch	—	✓	×
DA-RNN	×	✓	×
TPA-LSTM	×	✓	✓
SegRNN	✓	×	✓
PatchTST	✓	✓	✓
FSR (Ours)	✓	✓	✓

This distinction matters in LTSF because different future time steps may depend on predictive information from different parts of the historical context. Final-state readout requires this information to be consolidated into one representation supporting the entire forecast, whereas full-state readout permits horizon-specific combinations of representations from multiple temporal positions. A weaker recurrent forecast under the former interface therefore does not establish that recurrence is less capable of modeling temporal dependencies: it may also reflect the additional demand of terminal-state summarization. Without accounting for this demand, a restriction of the forecasting interface can be mistaken for a limitation of the encoder family.

The fixed-length bottleneck in sequence modeling motivated attention-based access to multiple encoder states (Bahdanau et al., 2015). Related recurrent forecasting studies have explored attention-based aggregation (Qin et al., 2017), while TPA-LSTM also evaluated hidden-state concatenation as an alternative (Shih et al., 2019). Building on these precedents, we systematically examine how representation accessibility changes recurrent–Transformer comparisons within an aligned LTSF pipeline. Specifically, we ask whether the apparent disadvantage of recurrent encoders persists when their intermediate states are directly accessible to the same type of non-autoregressive head used by strong patch-based baselines, without introducing attention-based aggregation or recurrent decoding. Table 1 summarizes the relevant differences in patch/segment inputs, multi-state readout, and non-autoregressive prediction.

To investigate this question, we introduce a controlled *Full-State Readout* (FSR) framework. FSR replaces the Transformer encoder with a standard recurrent encoder, such as vanilla RNN, GRU, or LSTM, while retaining normalization, channel independence, patching, positional encoding, and a Flatten–Linear readout form. Within this aligned forecasting pipeline, we systematically vary the number of recurrent hidden states exposed to the prediction head, from the final state to the complete state sequence. This design allows us to examine whether the apparent disadvantage of recurrent encoders persists when readout accessibility is aligned. On Traffic at a forecasting horizon of 192, switching the GRU-based model from final-state to full-state readout reduces MSE from 0.625 to 0.437, matching the reported MSE of full-state PatchTST. This example illustrates how a performance gap observed under different readout interfaces can disappear when representation accessibility is aligned. At the same forecasting horizon, full-state readout improves both MSE and MAE over final-state readout across all six datasets and all three recurrent backbones, showing that its benefits extend beyond a single dataset or recurrent cell.

Our contributions are summarized as follows:

- We identify *Representation Accessibility* as a confounding factor in recurrent–Transformer comparisons for LTSF, distinguishing the recurrent computation mechanism from the additional architectural constraint imposed by final-state readout.
- We introduce FSR to examine this factor within an aligned forecasting pipeline. By systematically varying the hidden-state readout scope, we provide evidence that representation accessibility can alter forecasting accuracy and the relative performance of recurrent and Transformer encoders.
- Across the evaluated benchmarks and recurrent backbones, we show that restricted final-state readout can underestimate the forecasting performance achievable with standard recurrent encoders. These findings support accounting for prediction interfaces before attributing performance gaps to encoder mechanisms.

2 Related Work

Transformer-based models have become a major paradigm for long-term time-series forecasting (LTSF), with representative methods improving forecasting performance through efficient attention, sequence decomposition, frequency-domain modeling, multi-scale representations, patching, and cross-variable interaction modeling (Zhou et al., 2021; Wu et al., 2021; Zhou et al., 2022; Liu et al., 2022a; Nie et al., 2023; Zhang & Yan, 2023; Li et al., 2019). However, strong LTSF performance is not exclusive to self-attention. DLinear shows that simple linear models can be competitive with Transformer-based forecasters (Zeng et al., 2023), while recurrent models such as LSTNet, DeepAR, and SegRNN remain effective for deterministic, probabilistic, and segment-wise forecasting (Lai et al., 2018; Salinas et al., 2020; Lin et al., 2026). Other neural forecasting approaches include NHITS, which combines multi-rate sampling with hierarchical interpolation (Challu et al., 2023); TiDE, which uses an MLP-based encoder–decoder and incorporates dynamic covariates (Das et al., 2023); TSMixer, which applies MLP-based mixing to patched time series and models cross-channel interactions (Ekambaram et al., 2023); and MTS-Mixers show that factorized temporal and channel mixing can model temporal dependencies without requiring attention (Li et al., 2025). These results suggest that observed performance gaps between forecasting architectures should not be attributed solely to the encoder backbone.

Recent studies further show that architectural components beyond the encoder can substantially affect forecasting accuracy. PatchTST demonstrates the importance of patching and channel-independent modeling (Nie et al., 2023); Crossformer and iTransformer adopt different channel or token organizations for multivariate forecasting (Zhang & Yan, 2023; Liu et al., 2024); and RevIN and Non-stationary Transformer highlight the influence of normalization, distribution shift, and non-stationarity handling (Kim et al., 2022; Liu et al., 2022b). These components may confound comparisons between recurrent and Transformer encoders, because models can differ not only in their sequence modeling mechanism but also in tokenization, channel structure, normalization, and prediction interface. We therefore keep patching, channel independence, normalization, positional encoding, and the Flatten–Linear head fixed as much as possible, and isolate the effects of encoder choice and representation accessibility.

Our work is most closely related to how recurrent representations are exposed to the prediction module. Recurrent models are often considered challenging for long sequences because information is propagated through sequential state updates, which may suffer from vanishing or exploding gradients (Bengio et al., 1994). LSTM and GRU mitigate this issue with gated transitions (Hochreiter & Schmidhuber, 1997; Cho et al., 2014), while recent segment-based recurrent forecasters reduce the number of recurrent steps by encoding local temporal segments (Lin et al., 2026). At the prediction-interface level, MQ-RNN combines recurrent encoding with a feed-forward decoder for direct multi-horizon quantile forecasting (Wen et al., 2017). Prior recurrent forecasting models such as DA-RNN and TPA-LSTM already use multiple hidden states through temporal or pattern-based attention (Qin et al., 2017; Shih et al., 2019), but they mainly study specific mechanisms for selecting or aggregating recurrent states. In contrast, we study *Representation Accessibility*: the range of encoder states directly accessible to a non-autoregressive forecasting head. Final-state readout restricts this range to the terminal recurrent state, whereas full-state readout exposes all recurrent encoder states. This allows us to test whether recurrent models are partly underestimated when compared with Transformers under restricted readout interfaces.

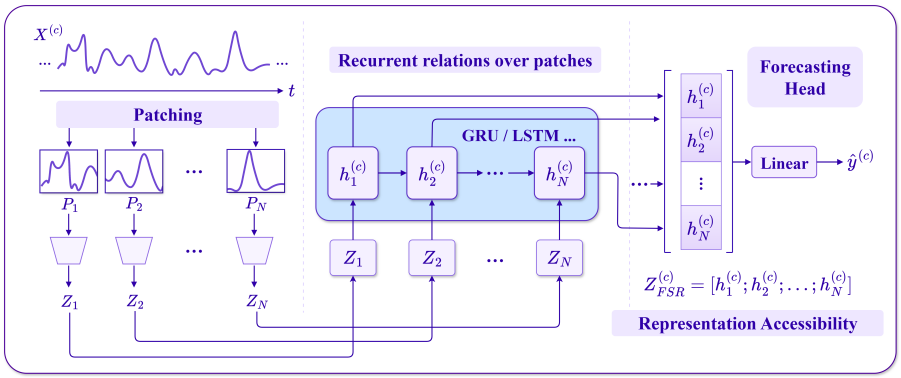


Figure 2: Overview of the Full-State Readout (FSR) framework. The input sequence is divided into overlapping patches and encoded by a recurrent encoder. Instead of using only the final hidden state, FSR exposes the full sequence of patch-level hidden states to a non-autoregressive Flatten–Linear forecasting head.

3 PROPOSED METHOD

3.1 PROBLEM SETUP AND PATCH-LEVEL INPUT REPRESENTATION

We consider multivariate long-term time-series forecasting (LTSF). Given a historical observation sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L] \in \mathbb{R}^{L \times C}$, where L is the input length, C is the number of variables, and $\mathbf{x}_t \in \mathbb{R}^C$ is the observation at time t, the goal is to predict the future sequence

$$\mathbf{Y} = [\mathbf{x}_{L+1}, \mathbf{x}_{L+2}, \ldots, \mathbf{x}_{L+T}] \in \mathbb{R}^{T \times C}, \tag{1}$$

where T is the forecasting horizon.

Following PatchTST (Nie et al., 2023), we adopt the channel-independent setting. Each variable is processed independently while sharing the same model parameters across channels. For the c-th variable, let

$$\mathbf{x}^{(c)} = [x_1^{(c)}, x_2^{(c)}, \ldots, x_L^{(c)}] \in \mathbb{R}^L \tag{2}$$

denote its historical sequence. The forecasting model learns a shared mapping $f : \mathbb{R}^L \to \mathbb{R}^T$ from each univariate input sequence to its future values. This setting allows us to focus on the sequence encoder and readout interface, rather than cross-variable dependency modeling.

Before encoding, each channel is normalized independently. We then apply replication padding of length S to the end of each normalized sequence and divide the padded sequence into $N = \lfloor (L+S-P)/S \rfloor + 1$ overlapping patches of length P and stride S, denoted as $\mathbf{P}^{(c)} = [\mathbf{p}_1^{(c)}, \mathbf{p}_2^{(c)}, \ldots, \mathbf{p}_N^{(c)}]$, where $\mathbf{p}_i^{(c)} \in \mathbb{R}^P$. When $S < P$, adjacent patches overlap along the temporal dimension. Unlike segment-based recurrent forecasters that use non-overlapping segments (Lin et al., 2026), we retain the overlapping patching scheme used in PatchTST to keep the input representation aligned with a strong Transformer-based baseline. Each patch is then projected into a d-dimensional latent space and combined with positional encoding, yielding the patch-token sequence

$$\mathbf{Z}^{(c)} = [\mathbf{z}_1^{(c)}, \mathbf{z}_2^{(c)}, \ldots, \mathbf{z}_N^{(c)}] \in \mathbb{R}^{N \times d}. \tag{3}$$

The same patching, embedding, positional encoding, normalization, and channel-independent processing are used for all encoder and readout variants in our comparisons.

3.2 FULL-STATE READOUT FRAMEWORK

Our goal is to isolate how the range of accessible encoder representations affects recurrent forecasting. To this end, we propose *Full-State Readout* (FSR), a controlled recurrent forecasting framework that replaces the Transformer encoder with a standard recurrent encoder while keeping the surrounding LTSF pipeline unchanged. The overall architecture of the FSR framework is illustrated

in Figure 2. FSR can be instantiated with different recurrent backbones, including vanilla RNN, GRU, and LSTM. Unless otherwise specified, we use the GRU instantiation as the default model in the main experiments.

Given the patch-token sequence $\mathbf{Z}^{(c)}$, the recurrent encoder updates its hidden state sequentially over patch positions:

$$\mathbf{h}_i^{(c)} = \mathcal{F}_\theta\left(\mathbf{z}_i^{(c)}, \mathbf{h}_{i-1}^{(c)}\right), \qquad i = 1, \ldots, N, \tag{4}$$

where $\mathbf{h}_i^{(c)} \in \mathbb{R}^{d_h}$ is the hidden state at the i-th patch position, d_h is the hidden-state dimension, and $\mathcal{F}_\theta$ denotes the recurrent transition function. Depending on the backbone, $\mathcal{F}_\theta$ is implemented as a standard RNN, GRU, or LSTM cell. Since we do not modify the internal recurrent units, we omit their standard transition and gating equations.

After recurrent encoding, the c-th variable yields a sequence of patch-level hidden states:

$$\mathbf{H}^{(c)} = [\mathbf{h}_1^{(c)}, \mathbf{h}_2^{(c)}, \ldots, \mathbf{h}_N^{(c)}] \in \mathbb{R}^{N \times d_h}. \tag{5}$$

Conventional recurrent forecasters often condition the prediction module only on the final hidden state $\mathbf{h}_N^{(c)}$. In contrast, FSR preserves all hidden states and exposes them jointly to the forecasting head. This design increases *Representation Accessibility*, i.e., the range of encoder representations directly available for prediction.

3.3 Non-Autoregressive Forecasting Head

To keep the prediction interface simple and comparable to patch-based Transformer forecasters, FSR uses a non-autoregressive Flatten–Linear head. Given the full hidden-state sequence $\mathbf{H}^{(c)}$, we flatten hidden states along the patch dimension:

$$\mathbf{r}^{(c)} = \mathrm{Flatten}\left(\mathbf{H}^{(c)}\right) = \left[(\mathbf{h}_1^{(c)})^\top, (\mathbf{h}_2^{(c)})^\top, \ldots, (\mathbf{h}_N^{(c)})^\top\right]^\top \in \mathbb{R}^{N d_h}. \tag{6}$$

Flattening preserves the ordered hidden-state sequence without introducing attention, pooling, or recurrent decoding. The forecasting head then maps the flattened representation directly to the entire prediction horizon:

$$\hat{\tilde{\mathbf{y}}}^{(c)} = \mathbf{W}_o \mathbf{r}^{(c)} + \mathbf{b}_o, \tag{7}$$

where $\mathbf{W}_o \in \mathbb{R}^{T \times (N d_h)}$, $\mathbf{b}_o \in \mathbb{R}^T$, and $\hat{\tilde{\mathbf{y}}}^{(c)} \in \mathbb{R}^T$ denotes the normalized prediction for the c-th variable. This head predicts all future steps in parallel and does not depend on previous predictions, avoiding autoregressive error accumulation.

Finally, the prediction is transformed back to the original scale using the inverse normalization statistics of the corresponding channel:

$$\hat{\mathbf{y}}^{(c)} = \mathcal{N}^{-1}\left(\hat{\tilde{\mathbf{y}}}^{(c)}\right) \in \mathbb{R}^T. \tag{8}$$

Predictions from all channels are recombined as $\hat{\mathbf{Y}} = [\hat{\mathbf{y}}^{(1)}, \hat{\mathbf{y}}^{(2)}, \ldots, \hat{\mathbf{y}}^{(C)}] \in \mathbb{R}^{T \times C}$.

Overall, FSR differs from PatchTST mainly in the sequence encoder: PatchTST generates patch-level representations using self-attention, whereas FSR generates them using recurrent state transitions. At the same time, FSR differs from conventional recurrent forecasters by making the full sequence of recurrent hidden states directly accessible to the same type of non-autoregressive Flatten–Linear head used in strong patch-based LTSF models.

4 Experiments

4.1 Experimental Settings

Datasets. We evaluate our method on six widely used long-term time-series forecasting benchmarks: ETTh1, ETTh2, ETTm1, ETTm2, Electricity, and Traffic. These datasets cover different sampling granularities and numbers of variables, providing diverse evaluation scenarios for long-term forecasting. Detailed dataset statistics are reported in Table 6 of Appendix A.1.

Baselines and metrics. We compare our method with 11 representative forecasting models from two architecture families. The recurrent baselines include SegRNN (Lin et al., 2026), LSTM (Hochreiter & Schmidhuber, 1997), GRU (Chung et al., 2014), and TPA-LSTM (Shih et al., 2019). The Transformer-based baselines include PatchTST (Nie et al., 2023), Autoformer (Wu et al., 2021), Crossformer (Zhang & Yan, 2023), FEDformer (Zhou et al., 2022), Informer (Zhou et al., 2021), Pyraformer (Liu et al., 2022a), and the vanilla Transformer (Vaswani et al., 2017). LSTM and GRU are included as classical recurrent baselines, while SegRNN represents a recent recurrent architecture designed for LTSF. Following standard practice, we report Mean Squared Error (MSE) and Mean Absolute Error (MAE), where lower values indicate better forecasting performance.

Implementation details. For Transformer-based models, we use three encoder layers, a model dimension of $d_{\text{model}} = 128$, 16 attention heads, and a feed-forward dimension of $d_{\text{ff}} = 256$. For recurrent models, we set the hidden-state dimension to 128 to maintain comparable representation capacity. Both PatchTST and the proposed FSR use a patch-based input representation with patch length 16 and stride 8. Instance normalization is applied to the input sequences. All models are trained with the Adam optimizer and a dropout rate of 0.2. The initial learning rate is 1×10^{-4} and is exponentially decayed by a factor of 0.5 after each epoch. The batch size is set to 16 for all datasets. Models are trained for up to 100 epochs with early stopping patience of 10. All experiments are implemented in PyTorch and conducted on a T-Head Zhenwu 810E PPU cluster.

4.2 Main Results

We first evaluate the overall forecasting performance of the proposed Full-State Readout framework under a unified experimental protocol. In this main comparison, we instantiate FSR with a GRU encoder, denoted as FSR-GRU, and use it as the representative variant for comparison with mainstream LTSF models. GRU is a standard and widely used recurrent backbone, making it a suitable choice for assessing whether a recurrent encoder with full-state readout can achieve competitive long-term forecasting performance. The effects of different recurrent instantiations, including RNN, GRU, and LSTM, are analyzed in subsequent controlled experiments.

Table 2: Average prediction results across prediction lengths 96, 192, 336, and 720. Best results are **bold**, and second-best results are underlined. Detailed results for each forecasting horizon are provided in Appendix A.2.

Architecture	Model	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
RNNs	FSR-GRU	**0.421**	**0.433**	**0.356**	**0.395**	**0.359**	**0.387**	**0.258**	**0.318**	**0.191**	**0.301**	**0.448**	0.329
	SegRNN	0.431	0.444	0.361	0.403	0.366	0.399	0.260	0.323	0.227	0.317	0.865	0.424
	LSTM	0.492	0.486	0.426	0.451	0.412	0.429	0.302	0.353	0.260	0.347	0.803	0.439
	GRU	0.482	0.481	0.421	0.449	0.410	0.426	0.307	0.355	0.260	0.347	0.803	0.442
	TPA-LSTM	1.066	0.817	2.970	1.478	0.801	0.679	1.458	0.928	0.370	0.438	0.854	0.469
Transformers	PatchTST	0.455	0.456	0.380	0.414	0.361	0.390	0.271	0.328	0.192	0.303	**0.448**	**0.328**
	Autoformer	0.510	0.499	0.466	0.494	0.517	0.496	0.480	0.426	0.277	0.378	0.735	0.454
	Crossformer	0.799	0.669	1.337	0.878	0.420	0.441	0.918	0.715	0.222	0.317	0.714	0.400
	FEDformer	0.520	0.507	0.424	0.459	0.433	0.450	0.321	0.373	0.291	0.389	0.699	0.442
	Informer	1.219	0.853	3.821	1.585	1.210	0.844	2.852	1.263	0.658	0.619	1.253	0.690
	Pyraformer	0.911	0.741	2.924	1.421	0.712	0.608	0.971	0.719	0.328	0.406	0.787	0.452
	Transformer	1.095	0.844	2.109	1.171	1.017	0.784	2.364	1.188	0.357	0.432	0.705	0.390

As shown in Table 2, FSR-GRU achieves the best or tied-best result on 11 of the 12 dataset–metric combinations and ranks second on the remaining one. It outperforms SegRNN on all 12 metrics, with clear gains on Electricity and Traffic, and surpasses PatchTST on 10 metrics.

These results show that recurrent models can remain highly competitive when combined with patch-based inputs and full-state readout. They further suggest that architectural comparisons depend not only on the encoder itself, but also on how encoded representations are made available for prediction. We next examine this factor through controlled experiments on representation accessibility.

Figure 3 further summarizes the relative performance of the twelve models using average ranks across the six datasets, where the ranking is computed from MSE averaged over four forecasting horizons. Following the Friedman test (Friedman, 1937), we perform Conover post hoc comparisons (Conover & Iman, 1979) with Holm correction (Holm, 1979) over all 66 model pairs. White

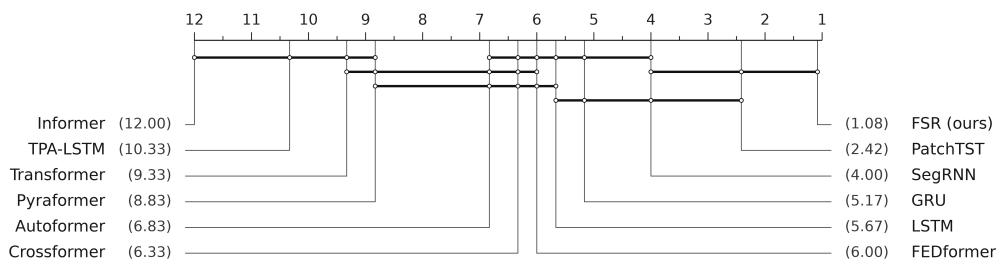


Figure 3: Average-rank diagram of 12 forecasting methods across six datasets. Lower ranks are better; thick bars indicate no significant difference under Conover–Holm tests ($\alpha = 0.05$).

circles connected by the same thick horizontal bar indicate methods for which no significant pairwise difference is detected under the corrected test.

FSR-GRU achieves the best average rank (1.08). Under the Conover–Holm procedure, its rank is not statistically distinguishable from those of PatchTST and SegRNN, while it is significantly better than the remaining nine baselines, including standard GRU and LSTM. These results indicate that FSR-GRU is among the strongest-performing methods under the evaluated protocol. This analysis complements the average-error comparison in Table 2, showing FSR-GRU's competitiveness is consistent at the rank level across datasets, rather than being driven by a single dataset or metric.

However, the rank analysis alone does not isolate the source of the improvement, since FSR-GRU differs from classical recurrent baselines in multiple components, including patch-based inputs and the forecasting head. Therefore, we use the following controlled experiments to examine whether the performance gain can be attributed to the proposed full-state readout and the resulting increase in Representation Accessibility.

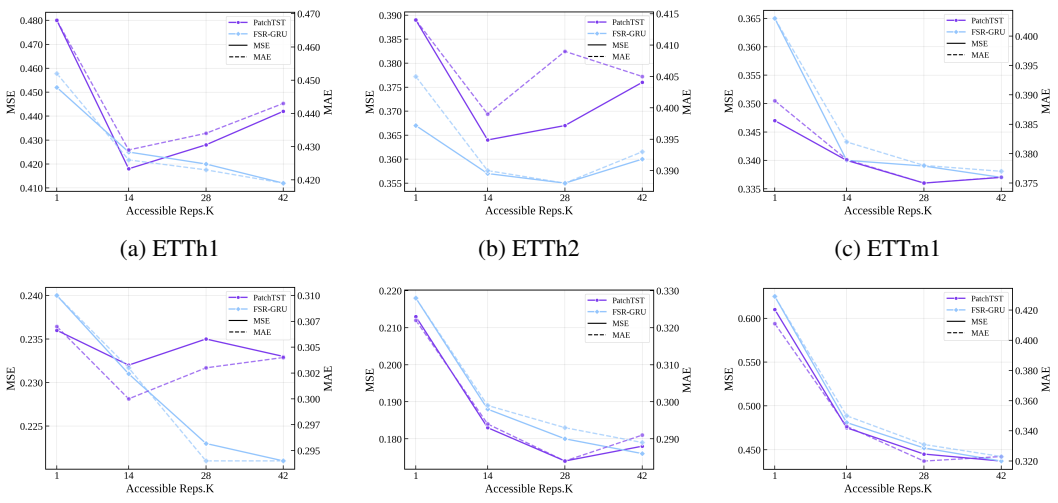
Table 3: Forecasting performance at prediction length 192 versus accessible representations K. FSR-GRU and PatchTST encode the same 42 patches, while the forecasting head receives the last K encoded representations.

Model	Accessible Reps. K	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
FSR-GRU	1	0.452	0.452	0.367	0.405	0.365	0.403	0.240	0.310	0.218	0.328	0.625	0.429
	14	0.425	0.426	0.357	0.390	0.340	0.382	0.231	0.303	0.188	0.299	0.481	0.350
	28	0.420	0.423	**0.355**	**0.388**	0.339	0.378	0.223	**0.294**	0.180	0.293	0.452	0.331
	42	**0.412**	**0.419**	0.360	0.393	0.337	0.377	**0.221**	**0.294**	0.176	0.289	**0.437**	0.323
PatchTST	1	0.480	0.468	0.389	0.414	0.347	0.389	0.236	0.307	0.213	0.322	0.610	0.411
	14	0.418	0.429	0.364	0.399	0.340	0.379	0.232	0.300	0.183	0.294	0.475	0.343
	28	0.428	0.434	0.367	0.409	**0.336**	**0.375**	0.235	0.303	**0.174**	**0.284**	0.445	**0.320**
	42	0.442	0.443	0.376	0.405	0.337	0.376	0.233	0.304	0.178	0.291	**0.437**	0.323
$\text{PatchTST}_{\text{Class}}$	[CLS]	0.524	0.494	0.396	0.422	0.347	0.387	0.238	0.307	0.211	0.320	0.630	0.425

4.3 How Does Representation Accessibility Affect Forecasting Performance?

To examine the effect of Representation Accessibility, we conduct a controlled experiment where the input sequences, encoder inputs, forecasting head, and training settings are fixed, while only the number of encoded representations accessible to the forecasting head is varied. Both FSR-GRU and PatchTST encode the same 42 input patches, and the forecasting head receives the last $K \in \{1, 14, 28, 42\}$ encoded representations. For FSR-GRU, these are GRU hidden states; for PatchTST, they are Transformer output tokens. Thus, $K = 1$ corresponds to final-state readout, while $K = 42$ corresponds to full-state readout. We also include $\text{PatchTST}_{\text{Class}}$, where a class token attends to all patch tokens but only the class-token output is used for forecasting, as a single-representation aggregation baseline.

Table 3 and Figure 4 show the results. For FSR-GRU, increasing K generally reduces forecasting errors, indicating that exposing more recurrent hidden states benefits prediction. PatchTST



(a) ETTh1 (b) ETTh2 (c) ETTm1

(d) ETTm2 (e) Electricity (f) Traffic

Figure 4: Forecasting performance of FSR-GRU and PatchTST as the number of representations K accessible to the forecasting head varies across six datasets.

also improves when K increases from 1 to 14 or 28, but further increasing K brings limited additional gains, likely because self-attention already allows each token to aggregate global information. PatchTST$_{\text{Class}}$ is consistently worse than multi-token PatchTST variants, suggesting that compressing the history into a single contextualized representation is insufficient.

The relative comparison between FSR-GRU and PatchTST changes substantially with K. When $K = 1$, FSR-GRU outperforms PatchTST on only 4 of the 12 dataset–metric combinations. When $K = 42$, it outperforms PatchTST on 8 metrics, matches it on 3, and is slightly worse only on the MAE of ETTm1. These results show that Representation Accessibility strongly affects architectural comparisons and support the view that final-state readout can restrict the usable information available from recurrent encoders.

Table 4: Prediction performance of recurrent encoders at a prediction length of 192 under final-state and full-state readout. For each architecture, *Final* uses only the last hidden state for forecasting, while *Full* provides all hidden states to the forecasting head.

Model	Setting	Metric	ETTh1	ETTh2	ETTm1	ETTm2	Electricity	Traffic
FSR-GRU	*Final*	MSE	0.452	0.367	0.365	0.240	0.218	0.625
		MAE	0.452	0.405	0.403	0.310	0.328	0.429
	Full	MSE	**0.412**	**0.360**	**0.337**	**0.221**	**0.176**	**0.437**
		MAE	**0.419**	**0.393**	**0.377**	**0.294**	**0.289**	**0.323**
FSR-LSTM	*Final*	MSE	0.457	0.370	0.368	0.250	0.222	0.638
		MAE	0.456	0.406	0.404	0.311	0.331	0.435
	Full	MSE	**0.422**	**0.353**	**0.336**	**0.227**	**0.174**	**0.434**
		MAE	**0.426**	**0.390**	**0.376**	**0.295**	**0.286**	**0.319**
FSR-RNN	*Final*	MSE	0.431	0.355	0.351	0.235	0.220	0.643
		MAE	0.438	0.390	0.390	0.307	0.328	0.428
	Full	MSE	**0.411**	**0.346**	**0.336**	**0.222**	**0.177**	**0.438**
		MAE	**0.419**	**0.384**	**0.373**	**0.294**	**0.290**	**0.324**

4.4 Does the Effect Generalize across Recurrent Architectures?

The preceding analysis shows that increasing the number of accessible hidden states improves the performance of FSR-GRU. We next examine whether this effect depends on the particular recurrent cell used by the encoder. To this end, we instantiate the FSR framework with three recurrent architectures: vanilla RNN, GRU, and LSTM. For each architecture, we compare two readout settings: *Final*, where the forecasting head receives only the last hidden state, and *Full*, where it receives the complete sequence of hidden states. All other components, including the input patching scheme, forecasting head, and training protocol, are kept unchanged.

Table 4 reports the results. Across all three recurrent architectures, full-state readout consistently improves both MSE and MAE over final-state readout on all six datasets. The gains are especially pronounced on Electricity and Traffic, where the input contains a larger number of variables and the forecasting task may require information distributed across multiple temporal patches. For example, on Traffic, full-state readout reduces the MSE from 0.625 to 0.437 for GRU, from 0.638 to 0.434 for LSTM, and from 0.643 to 0.438 for vanilla RNN. These results show that the benefit of full-state readout is not tied to the gating mechanism of a specific recurrent cell. Instead, the same pattern appears for gated recurrent encoders, such as GRU and LSTM, as well as for a vanilla RNN encoder. This supports the view that the final hidden state can act as an information bottleneck in recurrent forecasting models, and that exposing the full sequence of hidden states provides the forecasting head with more usable temporal representations for long-term prediction.

4.5 Computational Efficiency Analysis

In this section, we compare the computational cost of FSR-GRU with representative long-term time-series forecasting models. For the efficiency analysis, we use a unified single-layer configuration for all applicable models. We report three complementary efficiency measures: the number of trainable parameters, multiply–accumulate operations (MACs) estimated using THOP, and single-sample inference latency measured on the Electricity dataset. Parameters reflect model size, MACs provide an implementation-level estimate of arithmetic operations, and latency captures the realized inference time under the evaluated hardware and software configuration.

Table 5 presents the results. FSR-GRU has 1.134M parameters, 1.707G THOP-estimated MACs, and 2.970 ms inference latency. Compared with PatchTST, its patch-based Transformer counterpart, FSR-GRU has a similar parameter count and 20.5% fewer estimated MACs, but higher latency. This indicates that lower arithmetic cost does not necessarily lead to faster inference, likely because recurrent computation is less parallelizable than Transformer operations over patches.

Table 5: Efficiency comparison on Electricity.

Category	Model	Params (M)	MACs (G)	Time (ms)
RNNs	FSR-GRU	1.134	1.707	<u>2.970</u>
	SegRNN	**0.123**	2.140	3.240
	LSTM	8.181	0.086	7.830
	GRU	8.124	**0.066**	11.750
	TPA-LSTM	8.149	<u>0.067</u>	7.980
Transformers	PatchTST	1.167	2.147	**1.900**
	Autoformer	0.742	0.206	9.150
	Crossformer	3.013	6.399	11.860
	FEDformer	0.939	0.206	50.660
	Informer	<u>0.621</u>	0.177	6.860
	Pyraformer	24.001	0.126	5.690
	Transformer	<u>0.621</u>	0.177	5.950

Among recurrent models, FSR-GRU is also competitive. Compared with SegRNN, it has more parameters but 20.2% fewer MACs and slightly lower latency. Compared with conventional LSTM, GRU, and TPA-LSTM baselines, FSR-GRU has far fewer parameters and lower measured latency, although those baselines have smaller THOP-estimated MACs. Overall, FSR-GRU is not the lightest or fastest model in absolute terms, but it provides a favorable accuracy–efficiency trade-off under the evaluated configuration.

5 Conclusion

This work shows that the comparison between recurrent and Transformer-based forecasters is strongly influenced by how encoded representations are exposed to the forecasting head. We introduced the FSR framework to examine this factor by aligning key input components, including patching and channel independence, while varying Representation Accessibility under a common readout form. Across the evaluated datasets, recurrent architectures, and segmentation settings, our experiments show that expanding the readout scope generally improves recurrent forecasting performance and can substantially alter its relative comparison with Transformer encoders. These findings suggest that conventional final-state readout can restrict the usable information available from recurrent encoders in long-term forecasting. They also complement existing studies on patching and channel independence by highlighting Representation Accessibility as an additional factor in recurrent–Transformer comparisons. More broadly, our results indicate that sequence encoders should be evaluated not only by how they compute representations, but also by how those representations are made available for prediction. We hope this perspective encourages further investigation into recurrent forecasting and into the joint effects of input organization, encoding mechanisms, and readout design.

AI USE STATEMENT

We used generative AI for translation, polishing, and consistency checks. Generative AI was not used to develop the proposed methods, generate data, conduct experiments, or make theoretical claims. All AI-generated content has been rigorously reviewed by humans. The authors take full responsibility for the final content of this work.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. NHITS: Neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 6989–6997, 2023.

Kyunghyun Cho, Bart van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

W. J. Conover and R. L. Iman. On multiple-comparisons procedures. Technical report, Los Alamos Scientific Laboratory, 1979.

Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan K Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with TiDE: Time-series dense encoder. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Vijay Ekambaram, Arindam Jati, Nam Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. TSMixer: Lightweight MLP-Mixer model for multivariate time series forecasting. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 459–469, 2023.

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

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.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 95–104, 2018.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

Zhe Li, Xuanxuan Li, Zhongwen Rao, Lujia Pan, and Zenglin Xu. MTS-Mixers: Multivariate time series forecasting via factorized temporal and channel mixing. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2025.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

Shengsheng Lin, Weiwei Lin, Wentai Wu, Feiyu Zhao, Ruichao Mo, and Haotong Zhang. SegRNN: Segment recurrent neural network for long-term time-series forecasting. *IEEE Internet of Things Journal*, pp. 9861–9871, 2026.

Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X. Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2022a.

Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. *Advances in neural information processing systems*, 35: 9881–9893, 2022b.

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 *International conference on learning representations*, 2024.

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.

Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 2627–2633, 2017.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International journal of forecasting*, 36(3): 1181–1191, 2020.

Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. Temporal pattern attention for multivariate time series forecasting. *Machine Learning*, 108(8):1421–1441, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. In Edith Elkind (ed.), *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pp. 6778–6786. International Joint Conferences on Artificial Intelligence Organization, 2023.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems*, 34:22419–22430, 2021.

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

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The eleventh international conference on learning representations*, 2023.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11106–11115, 2021.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International conference on machine learning*, pp. 27268–27286. PMLR, 2022.

A Appendix

A.1 Dataset Details

Table 6 summarizes the number of variables, sampling granularity, and total number of time steps for each dataset. The four ETT datasets each contain seven variables, with ETTh1 and ETTh2 sampled hourly and ETTm1 and ETTm2 sampled every 15 minutes. Electricity and Traffic are hourly datasets containing 321 and 862 variables, respectively.

Table 6: Statistics of the six public forecasting datasets.

Dataset	ETTh1	ETTh2	ETTm1	ETTm2	Electricity	Traffic
Features	7	7	7	7	321	862
Granularity	1 hour	1 hour	15 mins	15 mins	1 hour	1 hour
Timesteps	17,420	17,420	69,680	69,680	26,304	17,544

A.2 Detailed Results across Forecasting Horizons

To complement the horizon-averaged comparison in the main text, Table 7 reports the MSE and MAE of all evaluated models on the six benchmark datasets at forecasting horizons $T \in \{96, 192, 336, 720\}$. These detailed results allow us to examine how the relative performance of the models varies across prediction lengths.

Across the 48 dataset–horizon–metric combinations, FSR-GRU achieves the best or tied-best result in 36 cases and ranks among the top two of the 12 evaluated models in 47, based on the reported numerical precision. The only result outside the top two is the MSE on ETTm1 at a forecasting horizon of 336. Averaged over the four horizons, FSR-GRU achieves the lowest MSE and MAE on all four ETT datasets. It also remains competitive at the longest forecasting horizon of 720, achieving the best or tied-best result in 11 of the 12 dataset–metric combinations. The relative performance nevertheless depends on the dataset and forecasting horizon. On Electricity, FSR-GRU achieves the best MSE and MAE at all four horizons, with PatchTST consistently ranking second. These results support the competitiveness of FSR-GRU across the evaluated forecasting horizons, while showing that its advantages are not uniform across datasets or prediction lengths.

A.3 Robustness of Representation Accessibility across Segmentation Granularities

The main-text analysis compares FSR-GRU and PatchTST with a fixed total of $N = 42$ input patches. Here, we extend the analysis to FSR-GRU with $N \in \{28, 84\}$ and SegRNN-based readout variants with $N \in \{7, 42\}$. We distinguish the total number of encoded patches or segments, N, from the number of representations accessible to the prediction module, K. Within each fixed-N configuration, we examine how forecasting performance changes as K increases. We first investigate whether the findings for FSR-GRU persist under additional segmentation settings, and then examine whether similar effects arise in SegRNN-based readout variants.

Table 8 and Figure 5 show that, for FSR-GRU, forecasting errors generally decrease as the number of accessible hidden representations K increases, under both the $N = 28$ and $N = 84$ segmentation settings. In both configurations, full-state readout ($K = N$) achieves lower MSE and MAE than final-state readout ($K = 1$) on all six datasets. However, the improvements are not strictly monotonic in K, and intermediate readout scopes yield the lowest errors in some cases. These results support the benefits of broader representation access beyond the default segmentation setting examined in the main text.

We next examine whether expanding the hidden-state readout scope also benefits SegRNN-based variants under different segmentation settings.

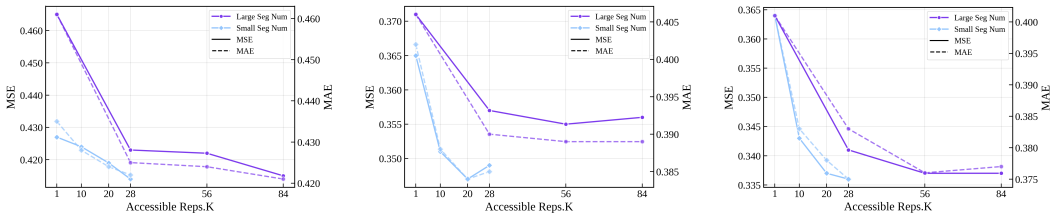
As shown in Table 9 and Figure 6, SegRNN-based readout variants also benefit from broader access to hidden representations. Compared with final-state readout ($K = 1$) within each fixed-N configuration, full-state readout reduces forecasting errors on all 12 dataset–metric combinations at

Table 7: Forecasting performance of all evaluated models at prediction lengths of 96, 192, 336, and 720. The best results are shown in **bold**, and the second-best results are underlined.

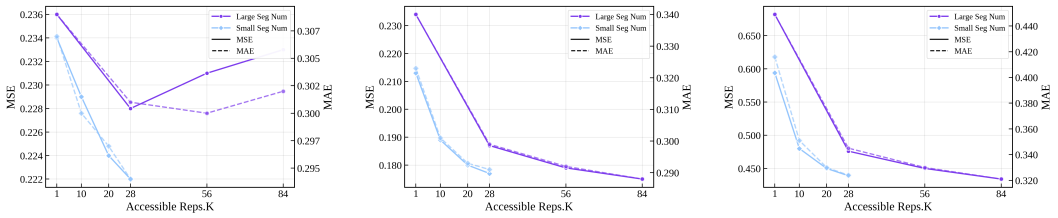
Pred len	Model	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
96	FSR-GRU	0.379	**0.399**	**0.285**	**0.342**	**0.298**	**0.349**	0.166	0.256	**0.160**	**0.276**	0.422	**0.316**
	SegRNN	0.389	0.415	**0.285**	0.346	0.313	0.367	**0.162**	**0.251**	0.198	0.293	0.838	0.418
	LSTM	0.451	0.460	0.425	0.446	0.377	0.410	0.206	0.292	0.240	0.332	0.745	0.423
	GRU	0.450	0.458	0.390	0.426	0.377	0.405	0.210	0.298	0.239	0.331	0.751	0.427
	TPA-LSTM	0.844	0.698	1.479	1.039	0.697	0.600	0.654	0.636	0.363	0.431	0.825	0.468
	PatchTST	**0.378**	0.403	0.315	0.367	0.300	0.355	0.175	0.265	0.161	0.277	**0.421**	**0.316**
	Autoformer	0.481	0.465	0.395	0.443	0.536	0.504	0.261	0.337	0.222	0.337	0.803	0.486
	Crossformer	0.511	0.500	0.574	0.550	0.308	0.357	0.297	0.370	0.189	0.288	0.660	0.382
	FEDformer	0.432	0.462	0.385	0.431	0.385	0.428	0.244	0.328	0.254	0.359	0.664	0.426
	Informer	1.077	0.781	3.147	1.404	0.748	0.619	1.087	0.841	0.414	0.475	1.002	0.566
	Pyraformer	0.650	0.586	1.536	1.029	0.577	0.522	0.439	0.513	0.334	0.407	0.788	0.465
	Transformer	0.960	0.788	1.602	1.001	0.764	0.649	1.169	0.855	0.327	0.412	0.754	0.424
192	FSR-GRU	**0.412**	**0.419**	0.360	0.393	**0.337**	0.377	**0.221**	**0.294**	**0.176**	**0.289**	**0.437**	**0.323**
	SegRNN	0.425	0.436	**0.350**	**0.388**	0.347	0.388	0.232	0.307	0.213	0.304	0.848	0.418
	LSTM	0.486	0.478	0.416	0.440	0.403	0.425	0.265	0.329	0.250	0.339	0.802	0.438
	GRU	0.467	0.468	0.425	0.448	0.388	0.417	0.272	0.332	0.249	0.339	0.796	0.440
	TPA-LSTM	1.059	0.812	3.298	1.568	0.730	0.639	0.837	0.707	0.367	0.435	0.846	0.464
	PatchTST	0.442	0.443	0.376	0.405	**0.337**	**0.376**	0.233	0.304	0.178	0.291	**0.437**	**0.323**
	Autoformer	0.498	0.477	0.419	0.462	0.504	0.496	0.296	0.359	0.355	0.428	0.709	0.445
	Crossformer	0.585	0.554	1.457	0.935	0.357	0.397	0.460	0.509	0.212	0.306	0.719	0.404
	FEDformer	0.518	0.501	0.414	0.452	0.410	0.439	0.291	0.354	0.284	0.384	0.698	0.443
	Informer	1.123	0.809	4.416	1.614	1.514	0.967	1.484	0.976	0.580	0.574	1.228	0.685
	Pyraformer	0.913	0.749	2.821	1.359	0.628	0.549	0.629	0.612	0.325	0.405	0.767	0.441
	Transformer	1.074	0.845	2.070	1.158	0.960	0.751	2.040	1.100	0.378	0.451	0.676	0.373
336	FSR-GRU	**0.437**	**0.440**	0.378	**0.409**	0.377	**0.397**	**0.279**	**0.333**	**0.193**	**0.304**	0.454	0.333
	SegRNN	0.447	0.449	**0.376**	0.417	0.373	0.406	0.283	0.341	0.230	0.320	0.868	0.424
	LSTM	0.520	0.501	0.420	0.451	0.415	0.432	0.329	0.372	0.261	0.348	0.818	0.441
	GRU	0.483	0.482	0.419	0.451	0.419	0.431	0.325	0.367	0.261	0.349	0.818	0.444
	TPA-LSTM	1.228	0.900	3.762	1.714	0.857	0.726	1.692	1.104	0.374	0.443	0.861	0.467
	PatchTST	0.456	0.451	0.393	0.425	**0.372**	0.398	0.291	0.341	0.195	0.306	**0.452**	**0.330**
	Autoformer	0.490	0.483	0.473	0.510	0.509	0.493	0.933	0.566	0.239	0.353	0.687	0.424
	Crossformer	1.032	0.791	1.387	0.877	0.447	0.463	0.958	0.771	0.229	0.325	0.740	0.412
	FEDformer	0.528	0.509	0.417	0.458	0.462	0.459	0.339	0.383	0.297	0.395	0.706	0.445
	Informer	1.223	0.849	3.117	1.432	1.218	0.880	2.134	1.152	0.676	0.630	1.239	0.676
	Pyraformer	1.021	0.799	3.911	1.668	0.730	0.631	0.991	0.749	0.324	0.406	0.785	0.445
	Transformer	1.102	0.819	2.072	1.162	1.148	0.866	2.531	1.244	0.365	0.440	0.690	0.383
720	FSR-GRU	**0.454**	**0.474**	**0.401**	**0.436**	**0.424**	**0.424**	0.367	**0.387**	**0.233**	**0.334**	**0.480**	**0.342**
	SegRNN	0.463	0.475	0.432	0.462	0.429	0.436	**0.364**	0.394	0.265	0.350	0.906	0.436
	LSTM	0.509	0.504	0.443	0.465	0.452	0.449	0.409	0.417	0.289	0.370	0.846	0.452
	GRU	0.526	0.515	0.449	0.470	0.455	0.451	0.421	0.422	0.290	0.370	0.848	0.456
	TPA-LSTM	1.132	0.857	3.341	1.590	0.919	0.749	2.647	1.265	0.376	0.443	0.883	0.476
	PatchTST	0.544	0.527	0.435	0.458	0.435	0.432	0.384	0.401	0.234	0.336	0.481	0.343
	Autoformer	0.572	0.569	0.576	0.562	0.518	0.492	0.429	0.441	0.292	0.394	0.740	0.460
	Crossformer	1.069	0.831	1.930	1.151	0.568	0.547	1.955	1.211	0.256	0.350	0.736	0.403
	FEDformer	0.601	0.557	0.479	0.494	0.475	0.473	0.410	0.425	0.330	0.418	0.727	0.454
	Informer	1.451	0.973	4.604	1.890	1.358	0.908	6.703	2.083	0.963	0.797	1.543	0.833
	Pyraformer	1.061	0.829	3.427	1.627	0.914	0.731	1.825	1.003	0.327	0.406	0.807	0.455
	Transformer	1.244	0.923	2.693	1.362	1.196	0.870	3.716	1.552	0.357	0.423	0.700	0.381

Table 8: Forecasting performance of FSR-GRU across six datasets with different numbers of segment-level hidden states K accessible to the forecasting head, with the prediction length fixed at $T = 192$. Best results are **bold**, and second-best results are underlined.

Model	Accessible Reps. K	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
FSR-GRU	1	0.427	0.435	0.365	0.402	0.364	0.401	0.234	0.307	0.213	0.323	0.594	0.416
	10	0.424	0.428	0.351	0.388	0.343	0.383	0.229	0.300	0.189	0.301	0.480	0.351
	20	0.419	0.424	**0.347**	**0.384**	0.337	0.378	0.224	0.297	0.180	0.293	0.450	0.330
	28	**0.414**	0.422	0.349	0.385	**0.336**	**0.375**	**0.222**	**0.294**	0.177	0.291	0.439	0.324
	1	0.465	0.461	0.371	0.406	0.364	0.401	0.236	0.309	0.234	0.340	0.682	0.449
	28	0.423	0.425	0.357	0.390	0.341	0.383	0.228	0.301	0.187	0.299	0.476	0.345
	56	0.422	0.424	0.355	0.389	0.337	0.376	0.231	0.300	0.179	0.292	0.450	0.330
	84	0.415	**0.421**	0.356	0.389	0.337	0.377	0.233	0.302	**0.175**	**0.288**	**0.434**	**0.321**



(a) ETTh1 (b) ETTh2 (c) ETTm1

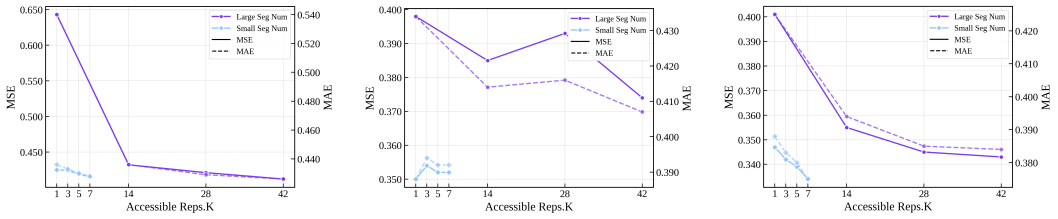


(d) ETTm2 (e) Electricity (f) Traffic

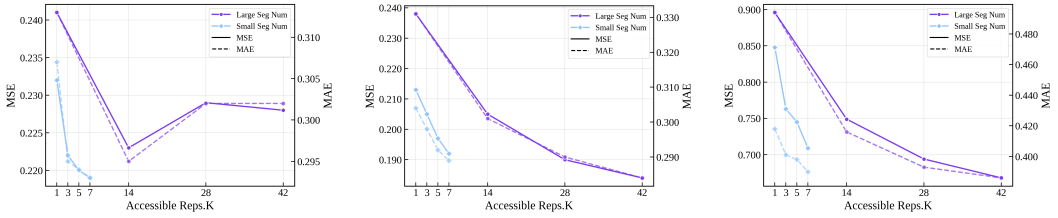
Figure 5: Forecasting performance of FSR-GRU under different total segmentation numbers across six datasets.

Table 9: Forecasting performance of SegRNN across six datasets with different numbers of segment-level hidden states K accessible to the forecasting head, with the prediction length fixed at $T = 192$. Best results are **bold**, and second-best results are underlined.

Model	Accessible Reps. K	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
SegRNN	1	0.425	0.436	**0.350**	**0.388**	0.347	0.388	0.232	0.307	0.213	0.304	0.848	0.418
	3	0.424	0.433	0.354	0.394	0.342	0.383	0.222	0.295	0.205	0.298	0.763	0.401
	5	0.419	0.430	0.352	0.392	0.339	0.380	0.220	0.294	0.197	0.292	0.745	0.398
	7	0.415	0.428	0.352	0.392	**0.334**	**0.375**	**0.219**	**0.293**	0.192	0.289	0.709	0.390
	1	0.643	0.540	0.398	0.434	0.401	0.425	0.241	0.313	0.238	0.331	0.896	0.494
	14	0.432	0.436	0.385	0.414	0.355	0.394	0.223	0.295	0.205	0.301	0.749	0.416
	28	0.421	0.429	0.393	0.416	0.345	0.385	0.229	0.302	0.190	0.290	0.694	0.393
	42	**0.412**	**0.426**	0.374	0.407	0.343	0.384	0.228	0.302	**0.184**	**0.284**	**0.668**	**0.386**



(a) ETTh1 (b) ETTh2 (c) ETTm1



(d) ETTm2 (e) Electricity (f) Traffic

Figure 6: Forecasting performance of SegRNN under different total segmentation numbers across six datasets.

$N = 42$ and on 10 of 12 at $N = 7$, with ETTh2 MSE and MAE being the exceptions. As with FSR-GRU, increasing K does not consistently improve every metric, and full-state readout is not always optimal. These findings suggest that the benefits of expanded representation access extend to another recurrent forecasting framework, while the preferred readout scope remains dependent on the dataset and segmentation setting.

A.4 Additional Encoder Variants within the FSR Framework

We further investigate whether modifying the encoder within the FSR framework yields consistent forecasting improvements. Specifically, we compare FSR-GRU with an echo state network variant, FSR-ESN, and their CNN-augmented counterparts, FSR-CNN-GRU and FSR-CNN-ESN. These comparisons examine the effects of replacing the recurrent backbone and incorporating additional feature extraction under full-state readout. Table 10 reports the results on six datasets.

Table 10: Forecasting performance of different FSR encoder variants on six datasets at $T = 192$.

Model	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
FSR-GRU	**0.412**	**0.419**	0.360	0.393	**0.337**	0.377	**0.221**	**0.294**	0.176	0.289	0.437	0.323
FSR-ESN	0.427	0.423	0.361	0.394	0.347	0.376	0.232	0.306	**0.169**	**0.274**	0.435	0.314
FSR-CNN-GRU	0.418	0.421	0.357	0.392	0.338	**0.374**	0.225	0.295	**0.169**	0.278	**0.423**	**0.305**
FSR-CNN-ESN	0.434	0.434	**0.356**	**0.389**	0.339	0.375	0.234	0.306	0.172	0.279	0.435	0.318

Table 10 shows that the effectiveness of combining different feature extraction and temporal modeling modules varies substantially across datasets. FSR-CNN-GRU achieves the best average forecasting performance, suggesting that CNN-based modeling of local relationships between adjacent patches complements the ability of GRU to capture long-term temporal dependencies. This advantage is particularly evident on Traffic and Electricity. In comparison, FSR-ESN performs well on Electricity and Traffic but degrades on several ETT datasets, indicating that the temporal representations produced by a fixed reservoir are sensitive to dataset characteristics. Adding a CNN enables FSR-CNN-ESN to achieve the best performance on ETTh2, but does not consistently improve results across all datasets, demonstrating that the effectiveness of combining CNN with ESN is also dataset-dependent. Overall, FSR-CNN-GRU provides more stable performance across datasets, while ESN-based architectures show potential as alternatives to trainable recurrent modules in specific settings.

A.5 Effect of Overlapping Patching on FSR-GRU

SegRNN uses non-overlapping segments (Lin et al., 2026), whereas FSR-GRU uses overlapping patches in the preceding experiments. Motivated by this difference, we compare overlapping and non-overlapping patching configurations within FSR-GRU while keeping the input length and number of patches fixed. Specifically, for an input sequence of 336 time steps, FSR-GRU uses patches of length 16 and stride 8, with replication padding of eight time steps at the sequence end. The non-overlapping variant, denoted as FSR-GRU_nooverlap, uses patches of length 8 and stride 8 without padding. Both configurations therefore provide 42 patch-level representations to the forecasting head.

Table 11: Forecasting performance of FSR-GRU with overlapping and non-overlapping patches at $T = 192$.

Model	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
FSR-GRU_overlap	0.412	0.419	0.360	0.393	0.337	0.377	0.221	0.294	0.176	0.289	0.437	0.323
FSR-GRU_nooverlap	0.416	0.423	0.352	0.387	0.331	0.373	0.221	0.294	0.181	0.296	0.444	0.331

Table 11 shows that overlapping patching performs better on ETTh1, Electricity, and Traffic, while the two settings achieve identical results on ETTm2. Non-overlapping patching performs slightly better on ETTh2 and ETTm1. Overall, the performance differences between the two settings are

modest. This result suggests that overlapping patching is not the main source of the performance gains achieved by FSR-GRU and may instead provide additional benefits on some datasets by preserving local temporal continuity.

A.6 Does Full-State Readout Require Sophisticated Aggregation?

To examine whether full-state readout requires a sophisticated aggregation mechanism, we keep the GRU encoder and input representation unchanged, provide each method with the same complete hidden-state sequence $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N]$, and vary only the readout strategy. FSR-GRU (Flatten) directly flattens all hidden states along the temporal dimension before applying a linear forecasting head. FSR-GRU (Cross-Attention) uses a learnable query to aggregate the hidden states into a single representation, whereas FSR-GRU (MeanPool) applies mean pooling. We evaluate these methods on six datasets with a forecasting horizon of $T = 192$ using MSE and MAE.

Table 12: FSR-GRU forecasting performance with different full-state readout strategies at $T = 192$.

Dataset	FSR-GRU (Flatten)		FSR-GRU (Cross-Attention)		FSR-GRU (MeanPool)	
	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	0.412	0.419	0.677	0.566	0.694	0.576
ETTh2	0.360	0.393	0.382	0.411	0.391	0.415
ETTm1	0.337	0.377	0.353	0.394	0.373	0.404
ETTm2	0.221	0.294	0.240	0.311	0.249	0.318
Electricity	0.176	0.289	0.236	0.341	0.253	0.352
Traffic	0.437	0.323	0.674	0.446	0.742	0.471

As can be seen from Table 12, FSR-GRU (Flatten) achieves the lowest MSE and MAE on all six datasets, with particularly large improvements on ETTh1, Electricity, and Traffic. Compared with Cross-Attention, Flatten reduces the MSE from 0.677 to 0.412 on ETTh1 and from 0.674 to 0.437 on Traffic. Although Cross-Attention generally outperforms MeanPool, it remains less effective than directly flattening the complete hidden-state sequence. These results suggest that the performance gains primarily arise from direct access to position-specific hidden representations rather than from additional aggregation or attention mechanisms.