TEMPODRAFT: EFFICIENT LONG-HORIZON ROLLING FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent time-series foundation models (TSFMs) achieve strong long-horizon forecasting performance, but their deployment for rolling forecasting remains costly. Their autoregressive decoding invokes the model sequentially for each forecast token. In autoregressive LLM generation, new tokens extend an unchanged prefix, allowing its KV cache to be reused across decoding steps. Rolling forecasting does not preserve this structure. Each new observation advances the fixed-length input window by one time step. This changes the input patches and their causal context, preventing direct reuse of the previous KV cache. However, the previous forecast still covers most timestamps in the new horizon. Thus, we introduce TEMPODRAFT, a training-free method that recasts rolling forecasting as a refinement of the previous forecast under new observations. This forecast provides a temporal self-draft for the next inference step. TEMPODRAFT first rebuilds the current context state and computes the first forecast token exactly as in autoregressive decoding. It then shifts the previous forecast to form an aligned draft and refines the remaining forecast tokens in parallel. A rolling forecast ensemble combines current and earlier forecasts of the same timestamps to stabilize predictions and improve accuracy. On Timer, extending the forecast horizon on hourly ETTh1 from approximately one month to one quarter increases the decoder speedup over autoregressive decoding from $2.3\times$ to $6.6\times$. At long horizons, TEMPODRAFT reduces MSE by 3.3% on ETTh1 and 6.6% on Electricity relative to the autoregressive baselines. Further experiments across nine benchmarks and multiple TSFM backbones support the method's generalizability and broad applicability.

1 INTRODUCTION

Long-horizon time-series forecasting in streaming applications requires continual revision of predictions (Pham et al., 2023; Lau et al., 2025). In energy systems, transportation, and environmental monitoring, new observations arrive continuously, and forecasts over a long future horizon must be updated in response (Zhou et al., 2021; Wu et al., 2022; Lim & Zohren, 2021). Recent time-series foundation models (TSFMs) have improved long-horizon forecasting (Liu et al., 2024b; Das et al., 2024; Shi et al., 2025; Liu et al., 2026), but they also increase the cost of these repeated updates. Advancing the input window by a single time step can trigger another complete pass of autoregressive (AR) decoding. Yet most timestamps in the new horizon were already predicted by the previous inference. Standard autoregressive pipelines discard those recent predictions and serially regenerate almost the same future (Liu et al., 2024b; Das et al., 2024; Liu et al., 2024a). This repeated computation becomes a major serving burden when long-horizon forecasts must be refreshed frequently.

As shown in Figure 1, adjacent rolling inference steps differ by only one new observation, and their forecast horizons overlap at all but one physical timestamp. This near-complete overlap suggests treating rolling inference as refinement of an existing forecast under the updated context. This resembles the motivation behind speculative decoding in LLMs. A smaller draft model proposes multiple candidate tokens, which the target model checks in parallel to reduce expensive serial steps (Chen et al., 2023; Li et al., 2024). Training and serving a separate draft model, however, adds overhead (Zhang et al., 2024) and leaves the previous TSFM forecast unused. Meanwhile, sliding the input window changes the causal context of retained observations, preventing direct reuse of the previous input tokens' key-value (KV) cache. Our goal is to reduce serial computation by refining the previous forecast under the current context while limiting degradation in forecast quality.

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

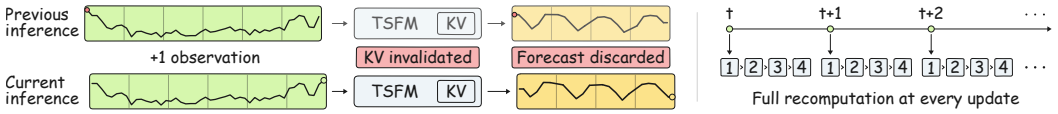

Figure 1: One-step updates repeat full inference. A single new observation shifts all input patch boundaries, invalidating direct input-KV reuse. Conventional rolling inference also discards the previous forecast and serially regenerates an H-point horizon with $H - 1$ overlapping timestamps.

To this end, we propose TEMPODRAFT, which implements this view with a training-free inference decoder that uses the previous forecast as a *temporal self-draft*. At each new inference step, TEMPODRAFT rebuilds the model state from the current context and computes the first forecast token exactly. This token anchors the current inference, and the time-aligned remainder of the previous forecast supplies provisional values for the later tokens. With causal attention, a single model call uses the current context and the previous-round draft to make the next forecast token match the output of AR decoding and correct the remaining draft in parallel. The extended matching prefix and corrected suffix then form the starting trajectory for the next round. Repeating this synchronous Jacobi-style update (Fu et al., 2024) advances the matching prefix one token at a time without serially regenerating the entire horizon. No auxiliary draft model or additional training is needed. After a chosen number of rounds, TEMPODRAFT's decoder produces the matching prefix together with an approximate suffix. At the same model-call budget, refining the reused suffix improves accuracy over copying it, showing the value of updating earlier predictions with new observations. Furthermore, forecasts from successive observation windows provide complementary information, motivating a rolling forecast ensemble that averages timestamp-aligned predictions without additional model calls. Although historical forecasts are often less accurate individually, averaging them with the current forecast can substantially reduce error.

We evaluate TEMPODRAFT on nine established forecasting benchmarks and multiple TSFM backbones. At a forecast horizon of 2304 steps, TEMPODRAFT achieves 2.4–6.6$\times$ speedups on ETTh1, ETTm1, Electricity, and PEMS04 while lowering MSE relative to AR decoding. On ETTm1, the unaveraged decoder lowers MSE by 2.7%. Rolling averaging over the full forecast horizon increases this reduction to 5.5%. For forecasts spanning approximately one quarter on hourly ETTh1, inference takes about 15.1% of the AR time and reduces MSE by 3.3%. Ablation studies show how temporal drafts, parallel refinement, and forecast averaging contribute to these gains. The main contributions are summarized as follows:

- We formulate rolling inference as a refinement of an existing forecast under new observations. Forecasts from successive observation windows provide temporal drafts for decoding and complementary predictions for output averaging.
- We introduce TEMPODRAFT, a training-free method that combines Jacobi-style parallel refinement with a rolling forecast ensemble. Each refinement round extends the prefix matching AR decoding by one token. The ensemble averages timestamp-aligned forecasts without additional model calls.
- Extensive experiments across nine benchmarks and four TSFM backbones show that TEMPODRAFT accelerates inference relative to AR decoding in all evaluated backbone–dataset settings while improving forecasting accuracy in most cases. Speedups and accuracy gains generally increase with the forecast horizon.

2 RELATED WORK

2.1 TIME-SERIES FOUNDATION MODELS

Large-scale pretraining enables time-series foundation models to make zero-shot forecasts on unseen datasets (Liu et al., 2024b; Das et al., 2024). A large proportion of these models use autoregressive token generation. Each token represents an individual observation or a segment of consecutive time points, and the model generates future tokens sequentially. Timer (Liu et al., 2024b) and TimesFM (Das et al., 2024) follow this basic design. Timer-XL (Liu et al., 2024a) extends next-token prediction to multivariate long-context forecasting. Chronos (Ansari et al., 2024) quantizes individual values

and generates future tokens autoregressively. Time-MoE (Shi et al., 2025) adds sparse experts, Toto (Cohen et al., 2025) strengthens cross-variate modeling, and Sundial (Liu et al., 2026) learns a generative distribution over the next token. Other approaches include the masked-encoder design of Moirai (Woo et al., 2024) and the periodical parallel decoder of LightGTS (Wang et al., 2025). Adler et al. (2026) study calibration over long autoregressive horizons, and Yu et al. (2026) analyze biases introduced by TSFM design choices. These studies address forecasting accuracy and generalization, whereas TEMPODRAFT focuses on the inference cost of existing TSFMs without training.

2.2 EFFICIENT TIME-SERIES FORECASTING

Early work on efficient time-series forecasting focused on reducing Transformer costs. Informer (Zhou et al., 2021) uses sparse attention, and Pyraformer (Liu et al., 2022) uses pyramidal attention. More recent designs include TQNet (Lin et al., 2025), which uses a single attention layer over temporal queries, and PENGUIN (Sun et al., 2026), which organizes periodic structure with grouped multi-query attention. Beyond attention-based designs, TimeMachine (Ahamed & Cheng, 2024) and Affirm (Wu et al., 2025) replace all-pairs attention with recurrent state updates, yielding linear or near-linear scaling. In addition, linear models such as FITS (Xu et al., 2024) and SparseTSF (Lin et al., 2024) reduce model size through frequency-domain interpolation and periodic sparsity, respectively. Lightweight TSFMs such as Tiny Time Mixers (Ekambaram et al., 2024) and LightGTS (Wang et al., 2025) likewise reduce parameter counts. Model simplification may trade accuracy or generalization for efficiency. TEMPODRAFT keeps the TSFM unchanged and accelerates inference through forecast reuse and parallel refinement.

2.3 SPECULATIVE AND PARALLEL DECODING

Autoregressive decoding generates one token at a time, so producing a complete output requires a sequence of serial model calls. Speculative decoding (Leviathan et al., 2023) is one of the main approaches to reducing these calls. A draft model first generates several candidate tokens, which the target model then verifies in parallel with a single forward pass. Subsequent work improves both draft generation and the execution pipeline. For draft generation, Lookahead Decoding (Fu et al., 2024) iteratively produces candidate n-grams, EAGLE (Li et al., 2024) constructs drafts from the target model's penultimate-layer features, and CLaSp (Chen et al., 2025) obtains a compact draft model by dynamically skipping layers of the target model. Fuzzy Speculative Decoding (Holsman et al., 2025) allows controlled deviations from the target-model distribution in exchange for greater speed. STRIDE (Subbaraman et al., 2026) applies speculative decoding to time-series forecasting with a lightweight draft model and parallel verification by the target model. TEMPODRAFT brings draft-based acceleration to rolling forecasting. It reuses the previous forecast as the draft for the next inference step, recomputes the first forecast token from the current context, and then extends the prefix that matches autoregressive decoding while refining the remaining draft tokens in parallel.

3 PRELIMINARIES

We consider TSFMs that generate forecasts autoregressively as sequences of tokens. At time t, the model uses a context of C observations $\mathbf{x}_{t-C+1:t} \in \mathbb{R}^{C\times D}$ to predict the next H time steps, where D is the number of channels. Each input token encodes a non-overlapping patch of consecutive observations (Liu et al., 2024b). In the decoding description below, a forecast token denotes the block of predicted time-series values associated with one autoregressive output position. Input and forecast tokens may span different numbers of time steps, depending on the model. An inference step produces the complete forecast for one observation window and may contain multiple model calls.

Let P denote the number of time steps in each forecast token. Covering the requested horizon requires $N = \lceil H/P \rceil$ tokens. Under causal attention, forecast token $\mathbf{z}_{t,i} \in \mathbb{R}^{P\times D}$ depends on the current context and earlier forecast tokens $\mathbf{z}_{t,1:i-1}$. We express forecast tokens in the model's normalized coordinates for the current observation window. Autoregressive decoding computes these tokens sequentially, appending each output to the sequence as a condition for later predictions. Concatenating the tokens and reversing this normalization gives the raw-unit forecast buffer $\mathbf{b}_t \in \mathbb{R}^{H_{\text{buf}}\times D}$. The basic construction uses $H_{\text{buf}} = NP$. The returned forecast is $\hat{\mathbf{y}}_t = \mathbf{b}_{t,1:H} \in \mathbb{R}^{H\times D}$. Within a single

inference step, the processed token prefix remains unchanged, allowing its KV cache to carry over to subsequent decoding steps.

We measure forecast quality by the mean squared error (MSE) between the predictions and the observations. With both expressed on the same evaluation scale, the error for inference step t is:

$$\mathrm{MSE}_t = \frac{1}{HD} \sum_{h=1}^{H} \sum_{d=1}^{D} \left(\hat{y}_{t,h,d} - x_{t+h,d}\right)^2 . \tag{1}$$

In rolling forecasting, each new observation triggers an update of the next H forecast steps. We study a stride of one observed time step. When $\mathbf{x}_{t+1}$ arrives, the input window and forecast horizon advance together:

$$\begin{aligned} \mathbf{x}_{t-C+1:t} &\longrightarrow \mathbf{x}_{t-C+2:t+1}, \\ \{t+1, \ldots, t+H\} &\longrightarrow \{t+2, \ldots, t+H+1\}. \end{aligned} \tag{2}$$

Adjacent input windows share $C-1$ observed timestamps, and adjacent forecast horizons share $H-1$ future timestamps. Most timestamps covered by the new forecast have already been predicted during the preceding inference.

This overlap does not translate directly into reuse under existing autoregressive decoding pipelines. Advancing the window removes the oldest observation from the causal context, and window-dependent normalization can change the representations of retained observations. A one-step shift also moves patch boundaries and changes the observations represented by each input token (Appendix F.2). Ordinary prefix reuse cannot generally recover the input-side KV cache for the updated window, so existing pipelines reprocess the largely overlapping context and autoregressively regenerate all N forecast tokens. Although conditioned on the previous context, old predictions remain aligned with most of the new horizon and can serve as an initial forecast for refinement.

4 TempoDraft

To reduce the cost of repeatedly forecasting overlapping horizons, TempoDraft reuses the previous forecast as an initial trajectory for the next inference step. We align it by target timestamp and refine it in parallel under the current observations, reducing sequential model calls with fixed model parameters and no additional training. We then apply a rolling forecast ensemble, averaging the refined forecast with earlier predictions of the same timestamps (Figure 2).

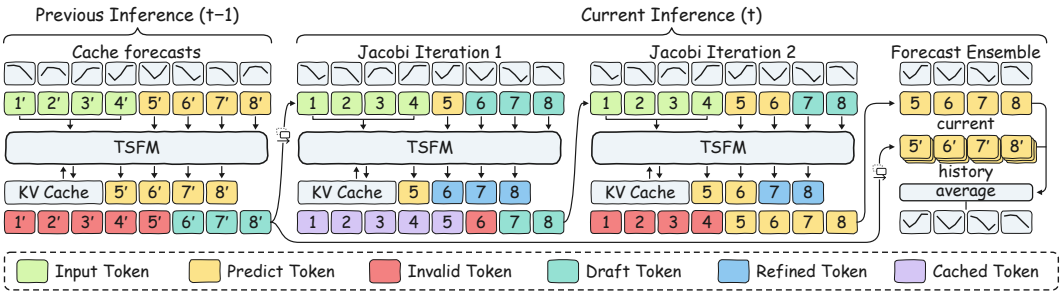

Figure 2: The TempoDraft decoder and rolling forecast ensemble. The previous forecast, aligned by target timestamp, initializes parallel refinement under the current observations. Numbers are token positions within an inference step, with forecast tokens at positions 5–8 and primes marking earlier steps. The first forecast token (position 5) is computed from the current context before the first refinement round. The ensemble at the right averages timestamp-aligned forecasts and its output never feeds back into the decoder.

4.1 Constructing the Temporal Self-Draft

To avoid the inference cost of a separate draft model, we reuse predictions already available from the preceding inference for most timestamps in the new horizon. Specifically, we align the saved raw-unit forecast with the current inference step by target timestamp, normalize the aligned values

using the current observation window, and repartition them into tokens to form the *temporal self-draft*. Appendix B gives the endpoint-filling rule. We encode the current observations $\mathbf{x}_{t-C+1:t}$ to obtain the context state $\mathcal{C}_t$ and generate the first forecast token. Writing $\mathbf{z}_{t,i}^{\mathrm{AR}}$ for normalized autoregressive token i under this context, we initialize $\mathbf{z}_t^{(0)}$ with $\mathbf{z}_{t,1}^{(0)} = \mathbf{z}_{t,1}^{\mathrm{AR}}$ and the draft values for tokens $2{:}N$. The first inference step uses autoregressive decoding because no previous forecast is available.

4.2 Parallel Refinement

Autoregressive decoding generates each token from the tokens already produced. With a draft available, we can instead update all later tokens in parallel. Each token uses the values from the previous round, so it need not wait for other tokens to finish their updates. This is a Jacobi iteration. The first token stays fixed, and the model's causal mask restricts each later token to the context and preceding tokens. We rebuild the context KV cache for every inference step and reuse it across refinement rounds. Let $\mathbf{z}_{t,i}^{(j)}$ be forecast token i after round j, and let g_i be the model's function for predicting that token, including embedding and positional encoding. For $0 \le J \le N-1$ refinement rounds, each round $j = 0, \dots, J-1$ applies:

$$\mathbf{z}_{t,1}^{(j+1)} = \mathbf{z}_{t,1}^{\mathrm{AR}}, \qquad \mathbf{z}_{t,i}^{(j+1)} = g_i(\mathcal{C}_t, \mathbf{z}_{t,1:i-1}^{(j)}), \qquad i = 2, \dots, N. \tag{3}$$

All token updates in a round are computed in one forward pass and applied together. Because the first token is fixed and attention is causal, each round extends the prefix matching AR decoding by one token. Before averaging, the first $J+1$ tokens therefore match AR decoding under the conditions in Appendix A. Both decoders use one forward pass to encode the current context and generate the first token. AR then makes $N-1$ sequential passes using its KV cache. TempoDraft makes J parallel refinement passes. The ratio of total model calls is therefore $N/(1+J)$. Actual speedup also depends on other runtime overhead (Appendix B). After refinement, we concatenate $\mathbf{z}_{t,1:N}^{(J)}$ and reverse the normalization to obtain $\mathbf{b}_t$, whose first H points form the forecast $\hat{\mathbf{y}}_t$ before averaging.

Suffix refinement within an inference step. Let $\varepsilon_{t,i}^{(j)} = \|\mathbf{z}_{t,i}^{(j)} - \mathbf{z}_{t,i}^{\mathrm{AR}}\|_F$ measure a normalized token's discrepancy from autoregressive decoding. Under a local blockwise Lipschitz bound for the causal maps g_i, the next-round discrepancy satisfies:

$$\varepsilon_{t,i}^{(j+1)} \le \sum_{\ell=1}^{i-1} B_{t,i\ell}\, \varepsilon_{t,\ell}^{(j)}, \qquad i = 2, \dots, N, \tag{4}$$

where $B_{t,i\ell} \ge 0$ bounds sensitivity to preceding token ℓ. This bound describes how discrepancies propagate along causal dependencies across refinement rounds. For fixed J, the resulting bound depends on both the initial draft discrepancy and the sensitivities along these paths. A temporal self-draft is useful when alignment keeps the initial discrepancy small. Causality makes the first $J+1$ tokens exact, but does not require every remaining suffix discrepancy to decrease at each round. Appendix A derives the conditional bound and distinguishes this agreement with autoregression from forecasting error against observations.

Recursive reuse and periodic refresh. Across inference steps, the refined forecast becomes the next draft, while real observations continue to determine the current context and first token. Before inserting the exact first token, the aligned draft's raw-unit discrepancy at a shared target timestamp, $h = 1, \dots, H-1$, decomposes as:

$$\hat{\mathbf{y}}_{t-1,h+1} - \hat{\mathbf{y}}_{t,h}^{\mathrm{AR}} = \big(\hat{\mathbf{y}}_{t-1,h+1} - \hat{\mathbf{y}}_{t-1,h+1}^{\mathrm{AR}}\big) + \big(\hat{\mathbf{y}}_{t-1,h+1}^{\mathrm{AR}} - \hat{\mathbf{y}}_{t,h}^{\mathrm{AR}}\big). \tag{5}$$

The first term carries residual approximation from the preceding inference. The second compares full AR predictions for the same future timestamp, made before and after the new observation enters the context window. These predictions can differ even if the preceding inference step used full AR. The updated AR forecast may be closer to or farther from the eventual observation. To interrupt the inherited chain, we generate a full autoregressive forecast every R inference steps, replacing the approximate trajectory with the current AR result. This limits uninterrupted reuse to at most $R-1$ approximate inference steps. Each non-refresh inference step still uses J refinement rounds: J controls computation within an inference step, and R controls how long residual discrepancies can pass between inference steps.

4.3 Rolling Forecast Ensemble

Ensembling is a common approach to improving forecasting accuracy, typically by combining predictions from different models (Mallet, 2010; Gao et al., 2023; Malhotra & Goel, 2025). In contrast, we combine forecasts produced by the same TSFM at different inference steps, averaging the current forecast with earlier predictions aligned to the same future timestamps. These forecasts use different observation windows and lead times, allowing errors that differ across inference steps to partially cancel. Appendix E.2 gives the corresponding squared-error decomposition.

We average the current forecast with up to $M-1$ previous forecasts, where $M \geq 1$. All predictions are first converted back to their original units. For time $t+h$, a forecast made a steps earlier supplies $\hat{\mathbf{y}}_{t-a,h+a}$, since $(t-a)+(h+a) = t+h$. Let $\mathcal{A}_{t,h}$ contain indices $a \in \{0, \ldots, M-1\}$ for which a forecast from inference step $t-a$ is available and $h+a \leq H$. The rolling forecast ensemble returns:

$$\hat{\mathbf{y}}^{\text{ens}}_{t,h} = \frac{1}{|\mathcal{A}_{t,h}|} \sum_{a \in \mathcal{A}_{t,h}} \hat{\mathbf{y}}_{t-a,h+a}, \qquad h = 1, \ldots, H. \tag{6}$$

At the start of a stream and near the end of the forecast horizon, fewer previous forecasts are available. We average only those that cover the target time, always including the current forecast. Setting $M = 1$ leaves it unchanged. Averaging covers the full horizon. We save predictions before averaging for both the next draft and the ensemble history. The averaged output is returned but never fed back into decoding. The ensemble requires no additional model call and can also be applied after ordinary autoregressive decoding. Uniform averaging yields nearly the same MSE as validation-selected aggregation, without fitting weights or introducing a decay parameter. Appendix E.3 compares fixed and learned aggregation rules.

5 Experiments

We evaluate the accuracy–latency tradeoff of temporal reuse and examine how refinement and forecast averaging contribute to it.

5.1 Experimental Setup

Datasets and models. We evaluate on nine widely used time-series forecasting benchmarks: ETTh1, ETTh2, ETTm1, and ETTm2 (Zhou et al., 2021), Weather, Electricity, and Traffic (Wu et al., 2022; Zeng et al., 2023), and PEMS04 and PEMS08 (Song et al., 2020). We use Timer (Liu et al., 2024b), whose forecast tokens span $P = 96$ time steps, as the main backbone and evaluate transfer to TimesFM 2.5 (Das et al., 2024), PatchFM (Vilhes, 2026), and Time-MoE (Shi et al., 2025). All models use publicly released pretrained weights and inference configurations (Appendix C.2). We compare TempoDraft against each backbone's AR baseline.

Evaluation. To reflect practical online forecasting, we update the forecast whenever a new observation arrives (Pham et al., 2023). Following the evaluation practices of TFB (Qiu et al., 2024) and TSLib (Wang et al., 2024), we use chronological data splits, standardize data using training-set statistics, and report MSE and MAE for fair comparisons. Unless noted otherwise, we use $H = 720$, $J = 2$ refinement rounds, ensemble size $M = 8$, and a full AR refresh every $R = 96$ inference steps. Speedup is the ratio of mean AR latency to mean TempoDraft latency. Backbone-specific settings and timing protocols are given in Appendices C and D.

5.2 Long-Horizon Accuracy and Efficiency

Main results. TempoDraft improves accuracy in 66 of 72 comparisons in Table 1 while accelerating all 36 backbone–dataset settings. The mean relative reductions are 2.1% for MSE and 1.2% for MAE, with all settings weighted equally. Specifically, for TimesFM, TempoDraft improves both MSE and MAE on all nine datasets with 1.4–$4.1\times$ speedups. For PatchFM, it reduces MSE by 13.0–15.7% on PEMS04 and PEMS08 at a $1.9\times$ speedup.

On Weather, TempoDraft increases MSE and MAE for Timer and Time-MoE. Zero-valued anomalies in multiple channels may make reused forecasts less reliable. Within each backbone, datasets

with similar channel counts tend to show similar speedups. Large channel counts already provide substantial parallel computation, leaving less room for additional gains from parallel refinement.

CUDA Graphs can further reduce kernel-submission overhead. For Timer on ETTh1 at $H = 720$, the CUDA Graph implementation reduces TEMPODRAFT's latency by 60.6% relative to its unoptimized implementation under idle conditions (Appendix D.3). Table 1 reports speedups without concurrent GPU workloads. Under concurrent matrix-multiplication workloads, TEMPODRAFT achieves larger speedups, reaching $12\times$ on Time-MoE and $13.3\times$ on TimesFM (Appendices D.2 and D.3).

Table 1: Forecasting performance of different backbones with and without TEMPODRAFT at $H = 720$. We report MSE, MAE, and inference speedup (Sp.); lower unrounded errors are **bold**.

Backbone	Timer					TimesFM 2.5					PatchFM					Time-MoE				
	AR		+Ours		Sp.	AR		+Ours		Sp.	AR		+Ours		Sp.	AR		+Ours		Sp.
Dataset	MSE	MAE	MSE	MAE	×	MSE	MAE	MSE	MAE	×	MSE	MAE	MSE	MAE	×	MSE	MAE	MSE	MAE	×
ETTh1	0.434	0.443	**0.427**	**0.439**	2.5	0.417	0.424	**0.415**	**0.423**	4.1	0.429	0.424	**0.429**	**0.423**	2.0	0.436	0.448	**0.431**	**0.447**	3.0
ETTh2	0.388	0.423	**0.387**	**0.422**	2.5	0.400	0.417	**0.399**	**0.417**	4.1	0.407	0.419	**0.403**	**0.418**	2.0	0.401	**0.431**	**0.401**	0.432	3.0
ETTm1	0.521	0.464	**0.512**	**0.460**	2.5	0.455	0.414	**0.449**	**0.412**	4.1	0.507	0.432	**0.498**	**0.429**	2.0	0.552	0.498	**0.535**	**0.489**	2.9
ETTm2	0.398	**0.409**	**0.397**	0.410	2.5	0.396	0.386	**0.393**	**0.385**	4.1	0.429	0.403	**0.424**	**0.401**	2.0	0.435	0.425	**0.427**	**0.421**	2.9
Weather	**0.359**	**0.367**	0.360	0.368	1.8	0.361	0.338	**0.360**	**0.338**	2.4	0.366	0.367	**0.348**	**0.340**	1.1	**0.356**	**0.366**	0.365	0.373	1.7
Electricity	0.221	0.307	**0.214**	**0.300**	1.6	0.201	0.282	**0.201**	**0.282**	1.4	0.211	0.290	**0.209**	**0.288**	1.5	0.230	0.325	**0.228**	**0.325**	1.4
Traffic	0.458	0.310	**0.445**	**0.295**	1.1	0.403	0.256	**0.402**	**0.255**	1.4	0.477	0.292	**0.471**	**0.288**	1.5	0.431	0.277	**0.421**	**0.273**	1.1
PEMS04	0.193	0.282	**0.190**	**0.280**	1.2	0.141	0.235	**0.140**	**0.235**	1.4	0.341	0.391	**0.297**	**0.367**	1.9	0.321	0.377	**0.302**	**0.367**	1.1
PEMS08	0.310	0.281	**0.307**	**0.279**	1.3	0.248	0.233	**0.244**	**0.232**	1.5	0.492	0.414	**0.415**	**0.388**	1.9	0.448	0.379	**0.429**	**0.370**	1.1

Accuracy and efficiency across horizons. Figure 3 shows that TEMPODRAFT generally provides larger accuracy gains and greater speedups as the forecast horizon increases. MSE is computed over the first H steps of 2304-step forecasts. Latency is measured separately at each H. On hourly ETTh1, extending the horizon from 8 to 24 tokens (768 to 2304 steps, approximately one month to one quarter) increases speedup from $2.3\times$ to $6.6\times$, and the MSE reduction grows from 1.9% to 3.3%. Longer horizons require more serial AR calls. In contrast, TEMPODRAFT keeps the number of refinement rounds fixed. Moreover, output averaging further improves accuracy on all four displayed datasets. Results for additional datasets and backbones are provided in Appendix D.1.

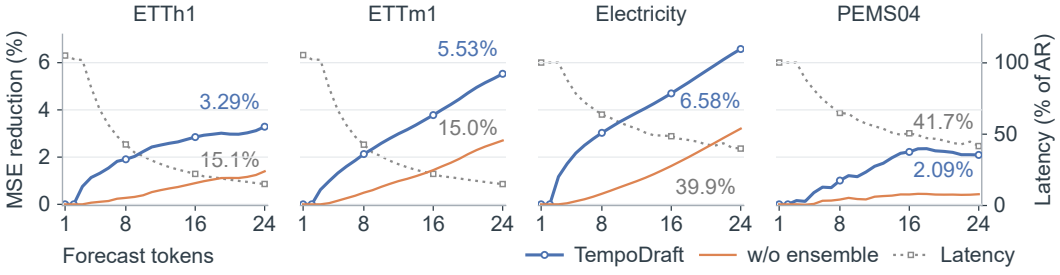

Figure 3: Cumulative MSE reduction (solid, left axis) and latency (dotted, right axis) relative to AR baseline on Timer backbone.

5.3 ABLATION STUDIES

Table 2 shows that TEMPODRAFT and AR with ensembling achieve the best overall forecasting accuracy, with closely matched results. The w/o ens. and w/o refine. rows use $M = 1$ and $J = 0$, respectively. Removing refinement increases MSE by 1.8–21.7% on all datasets. Replacing temporal drafts with zeros also degrades accuracy. Temporal drafts also better preserve AR outputs than other initializations (Appendix Table 12). Shift/paste generates the first three tokens with AR and copies the aligned previous forecast for the remaining horizon. With the same three model calls, refinement (w/o ens.) lowers MSE relative to Shift/paste on all eight datasets. In addition, ensembling is compatible with both TEMPODRAFT and AR, reducing MSE and improving forecast robustness. Compared with STRIDE (Subbaraman et al., 2026), which requires training an additional draft model, TEMPODRAFT

Table 2: Ablation study and baseline comparison using Timer as the backbone at $H = 720$. **Bold**: the best value, and underlined: the second-best value.

	ETTh1		ETTh2		ETTm1		ETTm2		Electricity		Traffic		PEMS04		PEMS08	
Method	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
TEMPODRAFT	0.427	0.439	**0.387**	**0.422**	0.512	0.460	**0.397**	0.410	0.214	**0.300**	**0.445**	**0.295**	**0.190**	**0.280**	**0.307**	**0.279**
w/o ens.	0.434	0.443	0.389	0.423	0.520	0.463	**0.397**	**0.409**	0.220	0.306	0.457	0.308	0.193	0.282	0.309	0.281
w/o refine.	0.435	0.448	0.398	0.426	0.525	0.455	0.405	0.415	0.260	0.329	0.492	0.331	0.214	0.294	0.333	0.290
w/o draft	0.497	0.496	0.397	0.433	0.577	0.497	0.404	0.416	0.383	0.441	0.653	0.433	0.413	0.453	0.515	0.461
AR	0.434	0.443	0.389	0.424	0.521	0.464	0.398	0.410	0.221	0.307	0.458	0.310	0.193	0.282	0.309	0.281
AR + ens.	**0.426**	**0.438**	0.388	0.423	0.513	0.461	0.398	0.410	**0.213**	**0.300**	**0.445**	**0.295**	**0.190**	**0.280**	**0.307**	**0.279**
Shift/paste	0.437	0.447	0.397	0.425	0.521	**0.454**	0.400	0.411	0.255	0.325	0.487	0.328	0.208	0.290	0.317	0.287
STRIDE	0.455	0.466	0.417	0.447	**0.496**	0.457	0.407	0.412	0.221	0.308	0.458	0.310	0.193	0.283	0.309	0.281

achieves a better overall accuracy. The ensemble reuses existing forecasts without additional model calls, and enabling it in TEMPODRAFT changes measured latency by at most 1.1%.

5.4 HYPERPARAMETER SENSITIVITY

Figure 4 compares ensemble sizes and $J \in \{1, 2, 3\}$ refinement rounds on four datasets, with additional results in Appendix E.1. An ensemble size around $M = 8$ offers a practical balance: larger ensembles bring only modest gains on ETTh1 and Electricity and can increase error on ETTm2. However, ETTm1 continues to benefit from longer histories. A second refinement round lowers MSE on all four datasets, even without ensembling ($M = 1$). On ETTm1, $J = 2$ with $M = 1$ already outperforms $J = 1$ with $M = 32$. The $J = 2$ and $J = 3$ curves nearly overlap, indicating that the benefits of refinement largely saturate after two rounds. These gains suggest that historical forecasts retain complementary predictive information that can improve the current forecast through averaging.

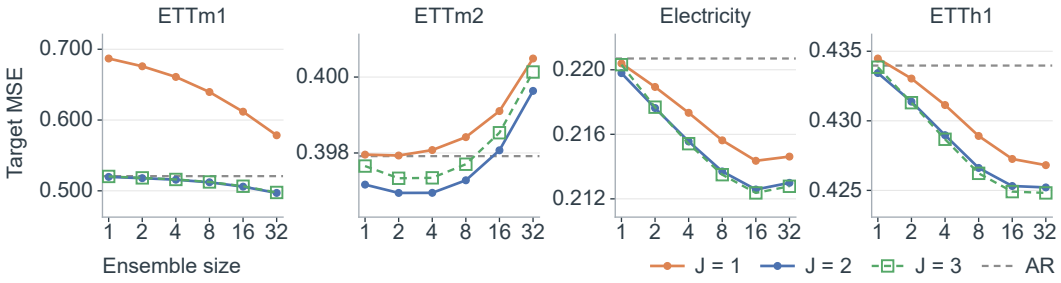

Figure 4: Effects of ensemble size M and refinement rounds J on MSE using Timer at $H = 720$.

5.5 ANALYSIS OF KV REUSE ACROSS INFERENCE STEPS

Reusing the input KV cache across inference steps could further accelerate TEMPODRAFT by avoiding repeated context encoding. StreamTimer (Meng et al., 2026) advances one patch at a time and retains historical KV entries within a fixed cache budget, reducing repeated context encoding. Our windows instead advance by one time step, changing patch contents and window normalization. The old KV cache therefore cannot be reused directly while preserving the recomputed forecast. Replacing old values with their exact current-window counterparts removes much more prediction deviation than replacing old keys in our Timer experiments. Keys determine attention weights, and values directly supply the content combined by those weights. On the ETTh1 dataset, excess forecasting error grows over consecutive cache-reuse steps and disappears after a full refresh. This pattern is consistent with approximation errors propagating through the reused state. TEMPODRAFT therefore rebuilds the input cache at each inference step and refines the reused forecast under the updated context. Appendix F.2 provides the component and rolling experiments.

5.6 Mechanistic and Qualitative Analysis

As shown in Figure 5, Linear CKA (Kornblith et al., 2019) reflects attention patterns becoming more similar to those of AR on ETTh1 and Traffic. Before refinement, attention in shallow layers is already close to AR, while deeper layers closer to the prediction show larger discrepancies. After two refinement rounds, attention patterns closely match AR across layers. Improvements beyond the exact prefix support recovery of AR conditioning at later forecast positions.

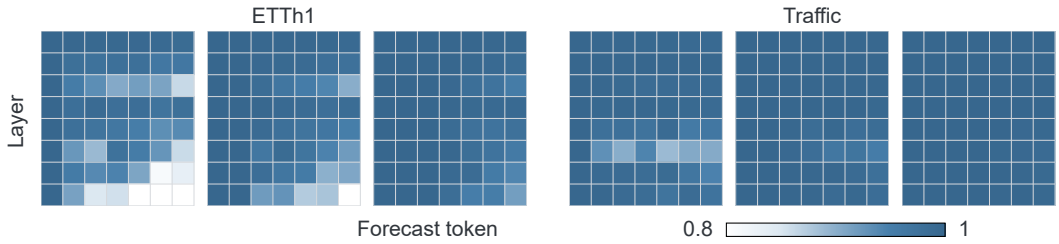

Figure 5: Attention CKA with AR after 0, 1, and 2 refinement rounds (left to right).

Figure 6 shows that individual historical forecasts are usually less accurate than the current forecast, but improve predictions over some intervals. Each cell reports the MSE change relative to the current forecast within a time bin; *Avg.* shows the change after averaging the current and historical forecasts. Averaging allows opposing errors to cancel while retaining corrections shared across forecasts.

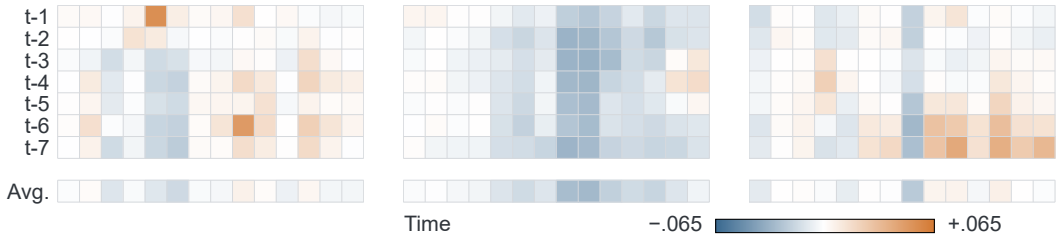

Figure 6: Changes in MSE relative to the current forecast (blue: improvement; orange: degradation).

Figure 7 presents local forecast comparisons on Electricity. In these examples, TempoDraft reduces overshoot and bias between peaks and better matches plateau levels and peak amplitudes than AR, while largely preserving the main turning points.

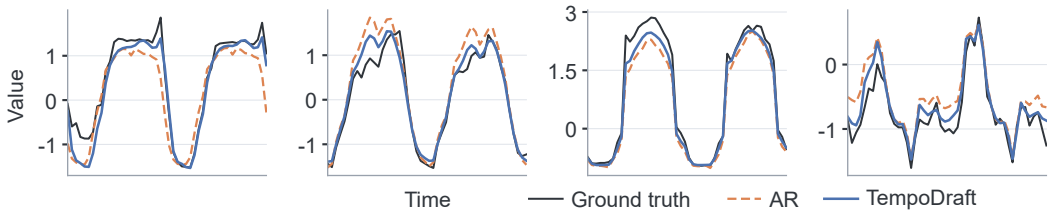

Figure 7: Forecasting examples of AR and TempoDraft on the Electricity dataset.

6 Conclusion

TempoDraft uses the previous forecast as a temporal self-draft for Jacobi-style parallel refinement and combines timestamp-aligned forecasts through a rolling ensemble. These two uses of prediction history reduce serial decoding and allow forecasts from different observation windows to complement one another. Across nine benchmarks and four TSFM backbones, TempoDraft accelerates inference in every backbone–dataset setting and improves forecasting accuracy in most of them.

Reproducibility Statement

We attach all the source code necessary to reproduce our experimental results in the supplementary materials. Appendix C documents the datasets, model configurations, rolling-evaluation protocol, and computational environment, while Appendix B provides implementation details.

Statement on Generative AI Use

Generative AI tools were used to assist with literature searches and manuscript polishing, and to accelerate code development and review. The authors reviewed and verified all AI-assisted content and take full responsibility for this work.

References

Coen Adler, Yuxin Chang, Samar Abdi, Felix Draxler, and Padhraic Smyth. Beyond accuracy: Are time series foundation models well-calibrated? In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 95913–95944, 2026.

Md Atik Ahamed and Qiang Cheng. Timemachine: A time series is worth 4 mambas for long-term forecasting. In *ECAI 2024*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pp. 1688–1695. IOS Press, 2024.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023.

Longze Chen, Renke Shan, Huiming Wang, Lu Wang, Ziqiang Liu, Run Luo, Jiawei Wang, Hamid Alinejad-Rokny, and Min Yang. CLaSp: In-context layer skip for self-speculative decoding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31608–31618, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0.

Ben Cohen, Emaad Khwaja, Youssef Doubli, Salahidine Lemaachi, Chris Lettieri, Charles Masson, Hugo Miccinilli, Elise Ramé, Qiqi Ren, Afshin Rostamizadeh, Jean du Terrail, Anna-Monica Toon, Kan Wang, Stephan Xie, Zongzhe Xu, Viktoriya Zhukova, David Asker, Ameet Talwalkar, and Othmane Abou-Amal. This time is different: An observability perspective on time series foundation models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50907–50951. Curran Associates, Inc., 2025.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10148–10167. PMLR, 21–27 Jul 2024.

Vijay Ekambaram, Arindam Jati, Nam H. Nguyen, Pankaj Dayama, Chandra Reddy, Wesley M. Gifford, and Jayant Kalagnanam. Tiny time mixers (ttms): Fast pre-trained models for enhanced zero/few-shot forecasting of multivariate time series. *CoRR*, abs/2401.03955, 2024.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the*

41st International Conference on Machine Learning, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 21–27 Jul 2024.

Yuanning Gao, Xiuqi Huang, Xuanhe Zhou, Xiaofeng Gao, Guoliang Li, and Guihai Chen. Dbaugur: An adversarial-based trend forecasting system for diversified workloads. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pp. 27–39, 2023.

Maximilian Holsman, Yukun Huang, and Bhuwan Dhingra. Fuzzy speculative decoding for a tunable accuracy-runtime tradeoff. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26257–26273, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 09–15 Jun 2019.

Ying-yee Ava Lau, Zhiwen Shao, and Dit-Yan Yeung. Fast and slow streams for online time series forecasting without information leakage. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 28736–28762, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 21–27 Jul 2024.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194): 20200209, February 2021. ISSN 1471-2962.

Shengsheng Lin, Weiwei Lin, Wentai Wu, Haojun Chen, and Junjie Yang. Sparsetsf: Modeling long-term time series forecasting with 1k parameters. In *Forty-first International Conference on Machine Learning*, 2024.

Shengsheng Lin, Haojun Chen, Haijie Wu, Chunyun Qiu, and Weiwei Lin. Temporal query network for efficient multivariate time series forecasting. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37797–37814. PMLR, 13–19 Jul 2025.

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*, 2022.

Yong Liu, Guo Qin, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer-xl: Long-context transformers for unified time series forecasting. *arXiv preprint arXiv:2410.04803*, 2024a.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. In *Forty-first International Conference on Machine Learning*, 2024b.

Yong Liu, Guo Qin, Zhiyuan Shi, Zhi Chen, Caiyin Yang, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Sundial: A family of highly capable time series foundation models, 2026.

Isha Malhotra and Nidhi Goel. Embracing the power of ensemble forecasting: A novel hybrid approach for advanced predictive modeling. *Information Processing & Management*, 62(1): 103954, 2025. ISSN 0306-4573.

Vivien Mallet. Ensemble forecast of analyses: Coupling data assimilation and sequential aggregation. *Journal of Geophysical Research: Atmospheres*, 115(D24), December 2010. ISSN 0148-0227.

Xiyu Meng, Yuhan Wu, Canran Xiao, Yabo Dong, and Duanqing Xu. Streamtimer: Efficient inference for long-context time series transformers. In Diego Calvanese (ed.), *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-26*, pp. 2942–2950. International Joint Conferences on Artificial Intelligence Organization, 8 2026. Main Track.

Quang Pham, Chenghao Liu, Doyen Sahoo, and Steven C. H. Hoi. Learning fast and slow for online time series forecasting. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S. Jensen, Zhenli Sheng, and Bin Yang. Tfb: Towards comprehensive and fair benchmarking of time series forecasting methods, 2024.

Xiaoming Shi, Shiyu Wang, Yuqi Nie, Dianqi Li, Zhou Ye, Qingsong Wen, and Ming Jin. Time-moe: Billion-scale time series foundation models with mixture of experts, 2025.

Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):914–921, Apr. 2020.

Pranav Subbaraman, Fang Sun, Jinxi Yu, Yue Yao, Huacong Tang, Xiao Luo, and Yizhou Sun. Accelerating time series foundation models with speculative decoding, 2026.

Tian Sun, Yuqi Chen, and Weiwei Sun. Penguin: Enhancing transformer with periodic-nested group attention for long-term time series forecasting. In Emtiyaz Khan, Yingzhen Li, Arno Solin, and Aaditya Ramdas (eds.), *Proceedings of The 29th International Conference on Artificial Intelligence and Statistics*, volume 300 of *Proceedings of Machine Learning Research*, pp. 703–711. PMLR, 02–05 May 2026.

Samy-Melwan Vilhes. PatchFM: A foundation model for univariate time series forecasting. GitHub software repository, 2026. Accessed September 22, 2026.

Yihang Wang, Yuying Qiu, Peng Chen, Yang Shu, Zhongwen Rao, Lujia Pan, Bin Yang, and Chenjuan Guo. LightGTS: A lightweight general time series forecasting model. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 64109–64126. PMLR, 13–19 Jul 2025.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Yong Liu, Mingsheng Long, and Jianmin Wang. Deep time series models: A comprehensive survey and benchmark. *CoRR*, abs/2407.13278, 2024.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53140–53164. PMLR, 21–27 Jul 2024.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, 2022.

Yuhan Wu, Xiyu Meng, Huajin Hu, Junru Zhang, Yabo Dong, and Dongming Lu. Affirm: Interactive mamba with adaptive fourier filters for long-term time series forecasting. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pp. 21599–21607. AAAI Press, 2025.

Zhijian Xu, Ailing Zeng, and Qiang Xu. Fits: Modeling time series with $10k$ parameters, 2024.

Annan Yu, Danielle Maddix, Boran Han, Xiyuan Zhang, Abdul Fatir Ansari, Oleksandr Shchur, Christos Faloutsos, Andrew Gordon Wilson, Michael W Mahoney, and Bernie Wang. Understanding the implicit biases of design choices for time series foundation models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 71998–72049, 2026.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In Brian Williams, Yiling Chen, and Jennifer Neville (eds.), *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pp. 11121–11128. AAAI Press, 2023.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

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 *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 11106–11115. AAAI Press, 2021.

A THEORETICAL ANALYSIS

Table 3: Notation used in the main text and theoretical analysis.

Symbol	Meaning
C, H, D	Context length, forecast length (in time steps), and channel count.
P, N	Time steps per forecast token and number of tokens covering the horizon, $N = \lceil H/P \rceil$.
H_{buf}, N_{buf}	Stored trajectory length and token count, $N_{\text{buf}} = H_{\text{buf}}/P \geq N$.
J, j	Refinement budget and round index; $j = 0$ denotes initialization.
R	Number of inference steps between full AR refreshes.
M	Maximum ensemble size, including the current forecast.
t, h, i	Inference time, forecast lead, and token index; lead h targets time $t + h$.
$\mathcal{C}_t$	Encoded context state for the current observation window.
$\mathbf{z}_{t,i}^{(j)}$, $\mathbf{z}_{t,i}^{\text{AR}}$	Refined and AR tokens in current-window normalized coordinates, each in $\mathbb{R}^{P\times D}$.
$\mathbf{b}_t$	Stored unaveraged forecast buffer in raw units, in $\mathbb{R}^{H_{\text{buf}}\times D}$.
$\hat{\mathbf{y}}_t$	Unaveraged forecast returned over the first H steps, $\mathbf{b}_{t,1:H} \in \mathbb{R}^{H\times D}$.
$\hat{\mathbf{y}}_t^{\text{ens}}$	Final timestamp-aligned ensemble forecast in raw units, in $\mathbb{R}^{H\times D}$.
L_{match}	Guaranteed AR-matching prefix length before averaging, $\min\{H, (J+1)P\}$.

A.1 NOTATION AND EXACT-PREFIX GROWTH

The context length, forecast length, and channel count are C, H, and D. An output token contains P points, so $N = \lceil H/P \rceil$ tokens cover the requested horizon. The stored trajectory may contain $N_{\text{buf}} = H_{\text{buf}}/P \geq N$ tokens. We write $\mathcal{C}_t$ for the current context state, $\mathbf{z}_{t,i}^{(j)}$ for normalized token i after refinement round j, and $\mathbf{z}_{t,i}^{\text{AR}}$ for its autoregressive reference. Saved forecasts use raw units. Evaluation scaling is applied equally to forecasts and targets.

Fix the current context, its preprocessing, and the positions for the current window. Assume that context encoding is independent of the draft and that sequential and parallel execution use the same deterministic, strictly causal maps:

$$\mathbf{z}_{t,1}^{\text{AR}} = g_1(\mathcal{C}_t), \qquad \mathbf{z}_{t,i}^{\text{AR}} = g_i(\mathcal{C}_t, \mathbf{z}_{t,1:i-1}^{\text{AR}}). \tag{7}$$

Parallel refinement fixes $\mathbf{z}_{t,1}^{(j)} = \mathbf{z}_{t,1}^{\text{AR}}$ and synchronously updates every later token from the previous round's prefix. Thus, for $i \geq 2$, $\mathbf{z}_{t,i}^{(j+1)} = g_i(\mathcal{C}_t, \mathbf{z}_{t,1:i-1}^{(j)})$.

Proposition 1 (Exact-prefix growth)**.** *Under these conditions, after* $0 \leq J \leq N_{\text{buf}} - 1$ *rounds:*

$$\mathbf{z}_{t,1:J+1}^{(J)} = \mathbf{z}_{t,1:J+1}^{\text{AR}} \quad \textit{in exact arithmetic}. \tag{8}$$

After truncation, at least $L_{\text{match}} = \min\{H, (J+1)P\}$ *returned points match AR before averaging.*

Proof. The first token is exact by construction. If tokens 1 through $j+1$ are exact after round j, each of the first $j+2$ outputs in the next round depends only on the same current context and an exact preceding prefix. It therefore equals the corresponding AR output. Induction proves the result. □

The result holds for any initializer. Causality determines prefix growth, while the temporal draft determines the quality of the remaining approximation under a fixed number of rounds. For the basic construction, $N_{\text{buf}} = N$. An extra stored token leaves the first N causal updates unchanged. In particular, $N - 1$ rounds recover the entire returned AR forecast, even when the stored trajectory extends beyond H. For Timer with $H = 720$ and $P = 96$, two rounds guarantee a 288-point matching prefix, and seven rounds recover all 720 returned points.

A.2 FINITE-BUDGET SUFFIX SENSITIVITY

Suppose a region containing the AR reference and refinement iterates admits nonnegative blockwise sensitivity bounds:

$$\|g_i(\mathcal{C}_t, \mathbf{u}_{1:i-1}) - g_i(\mathcal{C}_t, \mathbf{v}_{1:i-1})\|_F \leq \sum_{\ell<i} B_{t,i\ell} \|\mathbf{u}_\ell - \mathbf{v}_\ell\|_F, \tag{9}$$

where $\mathbf{B}_t$ is strictly lower triangular. For $\varepsilon_{t,i}^{(j)} = \|\mathbf{z}_{t,i}^{(j)} - \mathbf{z}_{t,i}^{\mathrm{AR}}\|_F$, substitution gives the componentwise bounds:

$$\boldsymbol{\varepsilon}_t^{(j+1)} \le \mathbf{B}_t \boldsymbol{\varepsilon}_t^{(j)}, \qquad \boldsymbol{\varepsilon}_t^{(J)} \le \mathbf{B}_t^J \boldsymbol{\varepsilon}_t^{(0)}. \tag{10}$$

The second bound follows by induction because $\mathbf{B}_t$ is nonnegative. Its powers sum sensitivity products along causal paths, separating the initial draft discrepancy from its propagation during refinement. Since the first entry of $\boldsymbol{\varepsilon}_t^{(0)}$ is zero, the first $J+1$ entries vanish after J rounds, recovering the prefix result. Strict lower triangularity gives $\mathbf{B}_t^{N_{\mathrm{buf}}} = 0$: causal dependence has a finite depth, so exact recovery does not require each round to be a contraction. At a smaller budget, the remaining entries depend on draft quality and path sensitivities, and their errors need not decrease monotonically.

A.3 Recursive Reuse and Refresh

At a shared target timestamp, adding and subtracting the previous AR forecast gives:

$$\begin{aligned}\hat{\mathbf{y}}_{t-1,h+1} - \hat{\mathbf{y}}_{t,h}^{\mathrm{AR}} = &\left(\hat{\mathbf{y}}_{t-1,h+1} - \hat{\mathbf{y}}_{t-1,h+1}^{\mathrm{AR}}\right) \\ &+ \left(\hat{\mathbf{y}}_{t-1,h+1}^{\mathrm{AR}} - \hat{\mathbf{y}}_{t,h}^{\mathrm{AR}}\right), \quad h = 1, \dots, H-1.\end{aligned} \tag{11}$$

The first term is inherited decoder discrepancy; the second is the change in the AR reference after new observations enter the context. Current-window normalization maps their sum to the coordinates of the initial draft, and replacing the first token sets its discrepancy to zero. A temporal draft starts close to the current reference when both terms are small relative to the current normalization scale.

An AR refresh sets the unaveraged decoder discrepancy to zero at that inference step. Refreshing every R steps therefore limits uninterrupted inheritance to at most $R-1$ approximate forecasts, while new observations can still change the next AR reference. J controls computation within an inference step, and R limits the length of uninterrupted recursive reuse.

A.4 AR Agreement and Forecasting Error

Let $\mathcal{I}_{\mathrm{tail}} = \{L_{\mathrm{match}} + 1, \dots, H\}$ be nonempty. On a common evaluation scale, write $\mathbf{e}_{t,h}^{\mathrm{AR}} = \hat{\mathbf{y}}_{t,h}^{\mathrm{AR}} - \mathbf{x}_{t+h}$ and $\boldsymbol{\delta}_{t,h} = \hat{\mathbf{y}}_{t,h} - \hat{\mathbf{y}}_{t,h}^{\mathrm{AR}}$. Expanding the squared error yields:

$$\mathrm{MSE}_{\mathrm{tail}}(\hat{\mathbf{y}}) - \mathrm{MSE}_{\mathrm{tail}}(\hat{\mathbf{y}}^{\mathrm{AR}}) = \frac{\sum_{h \in \mathcal{I}_{\mathrm{tail}}} \left(\|\boldsymbol{\delta}_{t,h}\|_2^2 + 2\langle \mathbf{e}_{t,h}^{\mathrm{AR}}, \boldsymbol{\delta}_{t,h} \rangle \right)}{|\mathcal{I}_{\mathrm{tail}}| D}. \tag{12}$$

Reference divergence measures the magnitude of $\boldsymbol{\delta}_{t,h}$. Forecasting error also depends on whether this deviation offsets or reinforces the AR error. Specifically, the refined forecast has lower tail MSE if and only if:

$$\sum_{h \in \mathcal{I}_{\mathrm{tail}}} \langle \mathbf{e}_{t,h}^{\mathrm{AR}}, \boldsymbol{\delta}_{t,h} \rangle < -\frac{1}{2} \sum_{h \in \mathcal{I}_{\mathrm{tail}}} \|\boldsymbol{\delta}_{t,h}\|_2^2. \tag{13}$$

The deviation must therefore oppose the AR error strongly enough to offset its own squared magnitude. This accounts for the different initializer rankings in Table 12 and for cases where approximate decoding improves target accuracy. The exact-prefix result applies to the unaveraged decoder. Full-horizon averaging can subsequently change that prefix.

B Implementation Details

B.1 State and Inference

The decoder saves the previous unaveraged trajectory $\mathbf{b}_{t-1}$ in raw units, including the complete final token. At stride one, it shifts this trajectory and repeats the last value at the new endpoint:

$$S(\mathbf{b}_{t-1}) = [\mathbf{b}_{t-1,2:H_{\mathrm{buf}}}, \mathbf{b}_{t-1,H_{\mathrm{buf}}}]. \tag{14}$$

For Timer with $H = 720$ and $H_{\mathrm{buf}} = 768$, this endpoint lies outside the returned horizon. Each reuse step recomputes the current-context KV, normalizes the shifted draft with the current window, inserts the exact first token, and performs J parallel rounds. The resulting unaveraged trajectory

supplies the next draft; its first H points enter the ensemble history. Averaged outputs are never fed back. Cold starts and scheduled refreshes use AR. Both histories reset at split boundaries, stream discontinuities, or changes to the model or inference settings; scheduled refreshes retain compatible ensemble history.

B.2 Runtime and Storage

A reuse step costs:

$$T_{\text{reuse}} = T_{\text{context}} + JT_{\text{refine}} + T_{\text{overhead}} + T_{\text{ens}}, \tag{15}$$

where the terms cover current-context encoding and the first token, parallel refinement, state handling, and averaging. In addition, stream latency includes scheduled AR refreshes. Timer with $H = 720$ and $J = 2$ uses three model calls per reuse step instead of eight AR calls. However, the measured speedup also depends on each call's workload and these overheads. Persistent storage is $O(H_{\text{buf}}D + MHD)$ for the draft and forecast history.

B.3 Empirical Prefix Agreement

The numerical results agree with Proposition 1. Across 4×721 Timer validation origins, the maximum absolute discrepancy from matched AR on the unaveraged prefix is 3.220×10^{-6}, with none exceeding 10^{-4}. The initialization controls likewise remain below this threshold, with a maximum of 1.910×10^{-6}. These comparisons use matched preprocessing and FP32 precision.

C Experimental Setup

C.1 Datasets

We evaluate on nine datasets. ETTh1, ETTh2, ETTm1, and ETTm2 contain electricity-transformer measurements; Weather records meteorological observations; Electricity measures power consumption; and Traffic, PEMS04, and PEMS08 provide road-occupancy or traffic-flow series. Table 4 summarizes their domains, sampling frequencies, and zero-based split boundaries: training $[0, t_{\text{train}})$, validation $[t_{\text{train}}, t_{\text{val}})$, and test $[t_{\text{val}}, t_{\text{test}})$. For PEMS04 and PEMS08, we use feature 0 (traffic flow) from the original NPZ arrays and a 60/20/20 split. The $H = 720$ comparison covers all 307 and 170 channels and all 2,680 and 2,853 valid test origins, respectively. For every dataset, channelwise scaling uses only the training mean and population standard deviation.

Table 4: Dataset domains, sampling frequencies, and chronological partitions.

Dataset	Domain	Frequency	t_{train}	t_{val}	t_{test}
ETTh1	Energy	1 hour	8,640	11,520	14,400
ETTh2	Energy	1 hour	8,640	11,520	14,400
ETTm1	Energy	15 min	34,560	46,080	57,600
ETTm2	Energy	15 min	34,560	46,080	57,600
Weather	Meteorology	10 min	36,887	42,156	52,696
Electricity	Energy	1 hour	18,412	21,044	26,304
Traffic	Transportation	1 hour	12,280	14,036	17,544
PEMS04	Transportation	5 min	10,195	13,593	16,992
PEMS08	Transportation	5 min	10,713	14,284	17,856

C.2 Foundation Models and Baselines

Table 5 lists context length C, refinement rounds J, and refresh period R for each backbone. Timer-base-84M is the primary model. TimesFM, PatchFM, and Time-MoE are used to evaluate transfer of the decoding procedure. All backbone weights remain fixed.

Table 5: Backbone configurations for $H = 720$.

Backbone	C	Native output geometry	J	R
Timer-base-84M	672	96-point tokens	2	96
TimesFM-2.5-200M	1024	128-point output blocks	1–2	96
PatchFM	1024	32-point patches	2	96
Time-MoE-50M	512	64-point heads	2	96

Baseline comparisons. For each backbone, TEMPODRAFT and its AR baseline use the same pretrained weights, context and forecast lengths, preprocessing, and numerical precision. The AR baseline uses serial decoding under the same backbone configuration. All compared methods, including the AR baseline, are scored on the same forecast origins within each comparison. AR with an ensemble uses the same $M = 8$ averaging as TEMPODRAFT. Shift-and-paste generates the first three tokens with AR and fills the remaining horizon with the previous forecast aligned to the current timestamps. We adapt STRIDE (Subbaraman et al., 2026) to the same frozen Timer-base target and train its draft on each dataset's training split. Checkpoints, draft lengths, and acceptance thresholds are selected on validation data before testing. Within each forecast, verification reuses the context KV cache and the accepted forecast prefix shared by active channels.

C.3 INFERENCE CONFIGURATION AND EVALUATION

Table 6 summarizes the main inference settings. Timer and TimesFM retain 768-point buffers at $H = 720$. PatchFM retains 736 points and returns the first 720. Time-MoE uses a 64-point serial anchor followed by two parallel refinement rounds with its native forecast heads and memory-efficient SDPA. On Electricity and Traffic, TimesFM generates two output tokens serially and then applies one parallel refinement round; its other backbone comparisons use two rounds.

Table 6: Inference settings and hyperparameter ranges.

Setting	Configuration
Forecast horizon	$H = 720$; additional long-horizon results in Appendix D
Rolling stride	One newly observed time point
Refinement rounds	$J = 2$; sensitivity to $J \in \{1, 2, 3\}$
Output averaging	Uniform mean, $M = 8$; current-only output uses $M = 1$
Ensemble size	$M \in \{1, 2, 4, 8, 16, 32\}$
Refresh period	Full AR every $R = 96$ inference steps
Component removals	$M = 1$; $J = 0$ with the first token retained; or zero draft with $J = 2$

Rolling evaluation. We use the standard MSE and MAE, averaged over forecast origins, prediction steps, and channels after training-split scaling. On the same set of forecast origins, this scoring is equivalent to standard sliding-window evaluation: processing windows chronologically changes the inference schedule, not the metric definition or weighting. The input window advances by one newly observed point, and forecasts are updated using only information available at that time, as in practical deployment. An origin is valid only if its entire target horizon falls within the evaluation split; previously observed context may precede that boundary. Each decoder maintains its own unaveraged state and forecast history, initialized empty at the start of the split. Averaging uses only available forecasts for the same target timestamp and never feeds back into decoding. The backbone comparison and ensemble size sweep retain all valid test origins, including cold start. The component and fixed-rule comparisons exclude the same first seven origins for every method. Decoder divergence from AR is evaluated separately, before averaging.

C.4 COMPUTATIONAL ENVIRONMENT

The RTX 3090 Ti platform has one NVIDIA GeForce RTX 3090 Ti with 24 GB GPU memory, an Intel Core i9-12900KF processor (16 cores, 24 threads), and approximately 94 GiB of system memory.

It runs Ubuntu 20.04.6 LTS with Linux kernel 5.15.0-105-generic and NVIDIA driver 535.161.08. The software environment uses Python 3.10.15, PyTorch 2.2.2 with CUDA 12.1, Transformers 4.40.1, NumPy 1.26.0, and Triton 2.2.0. Inference uses FP32 with TF32 disabled, and timing runs use one CPU thread per inference process.

D Additional Forecasting and Efficiency Results

D.1 Additional Forecast-Length Results

Figure 8 compares forecast lengths on ETTh2, ETTm2, Weather, and PEMS04, using all valid test origins for $H = 2304$. As in Figure 3, accuracy curves summarize cumulative MSE over the first H steps of forecasts produced with $J = 2$. Latency is measured separately at each requested H. Output averaging improves on the unaveraged decoder at long horizons on all four datasets. At 2304 steps, TempoDraft reduces MSE relative to AR by 1.2% on ETTh2 and 2.1% on PEMS04 but increases it by 0.27% on ETTm2 and 1.5% on Weather.

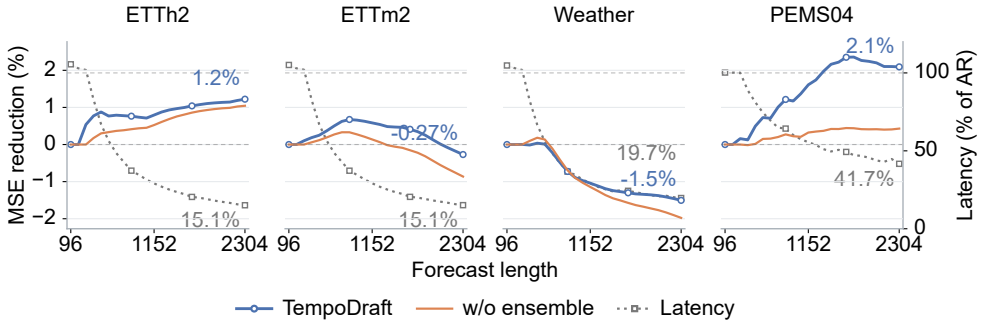

Figure 8: Cumulative MSE reduction and relative latency for Timer on four datasets.

Figure 9 further extends Figure 3 to TimesFM, PatchFM, and Time-MoE on ETTh1. The accuracy curves use the same setting. Output averaging improves on the unaveraged decoder for all three backbones. At $H = 2304$, TempoDraft reduces MSE relative to AR by 1.2% for TimesFM and 1.8% for Time-MoE, with a 0.21% increase for PatchFM; the corresponding speedups are $6.6\times$, $3.6\times$, and $2.8\times$.

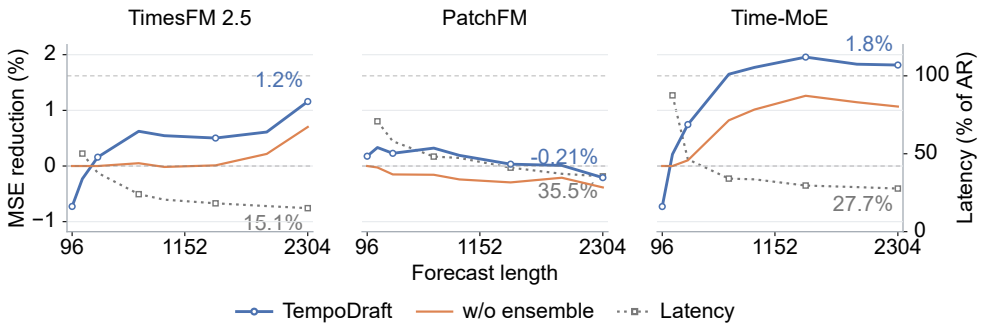

Figure 9: Cumulative MSE reduction and relative latency for three additional backbones on ETTh1.

Figure 10 reports the same comparison on ETTh2, using all 577 complete test windows at $H = 2304$. At this horizon, TimesFM, PatchFM, and Time-MoE achieve speedups of $7.0\times$, $2.8\times$, and $3.7\times$, respectively.

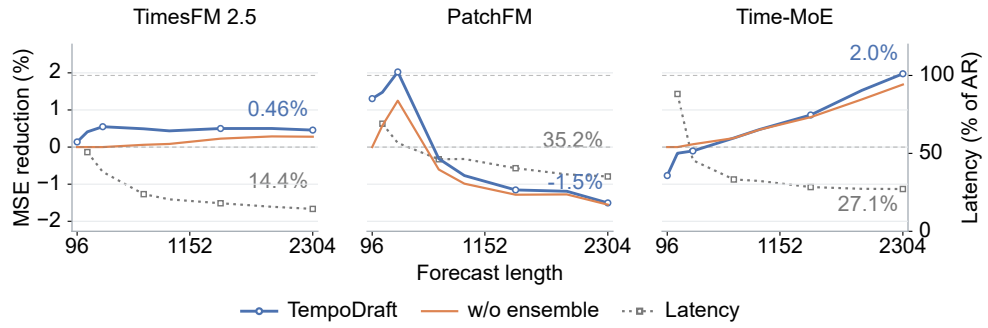

Figure 10: Cumulative MSE reduction and relative latency for three backbones on ETTh2.

Figure 11 extends this comparison to ETTm1, using all 9,217 complete test windows at $H = 2304$. At this horizon, TimesFM, PatchFM, and Time-MoE achieve speedups of $6.7\times$, $2.8\times$, and $3.5\times$, respectively. TEMPODRAFT reduces MSE relative to AR by 3.6% for TimesFM and 9.8% for Time-MoE, with an 8.1% increase for PatchFM.

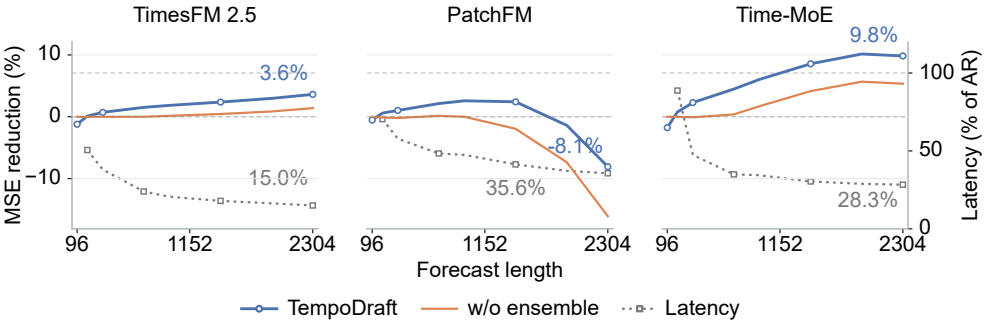

Figure 11: Cumulative MSE reduction and relative latency for three backbones on ETTm1.

Confidence intervals. For the Timer comparison at $H = 720$ in Table 2, we compute 95% confidence intervals using paired circular block bootstrap with 10,000 resamples. Block lengths are 71, 230, and 145 for ETTh1, ETTm1, and Electricity, respectively. The intervals for both MSE and MAE reductions are above zero on all three datasets.

Table 7: Relative error reduction (%) over AR with 95% confidence intervals.

Dataset	MSE	MAE
ETTh1	1.69 [1.43, 1.94]	0.97 [0.84, 1.10]
ETTm1	1.67 [1.18, 2.21]	0.85 [0.68, 1.03]
Electricity	3.17 [2.94, 3.43]	2.50 [2.29, 2.71]

D.2 LATENCY UNDER CONCURRENT GPU WORKLOADS

We evaluate all four backbones on ETTh1 at $H = 720$ using one RTX 3090 Ti, seven channels, FP32, and disabled TF32. The backbone-specific context lengths and two-round decoders are unchanged, with $R = 96$ and uniform $M = 8$.

A separate process repeatedly multiplies 2048×2048 FP32 matrices, synchronizing after each multiplication. Each backbone is measured on two independent 96-step validation streams, with empty initial histories, eight excluded training warm-ups per condition, and alternating method order. Each stream includes idle measurements before and after the two loaded conditions. The second stream reverses the order of the loaded conditions. Timing covers CPU input through CPU output, including refreshes and output averaging, and retains every sample.

Table 8: End-to-end speedup over AR under concurrent GPU workloads on ETTh1 ($H = 720$).

Backbone	Idle	One process	Two processes
Timer	2.5	2.5	2.4
TimesFM 2.5	4.1	10.7	13.3
PatchFM	2.0	2.4	2.7
Time-MoE	3.0	10.3	12.0

Table 8 shows that the response to GPU workloads depends on the backbone.

D.3 Implementation Optimizations

TempoDraft uses CUDA Graphs to reduce kernel-submission overhead for Timer and Time-MoE. TimesFM additionally keeps cache positions on the CPU, removes diagnostic synchronizations, and skips the unused quantile-spread output head. PatchFM avoids redundant cache expansion and reuses prepared context keys and values across refinement rounds. These changes preserve the specified decoders and uniform $M = 8$ outputs.

We evaluate ETTh1 at $H = 720$ on an RTX 3090 Ti with the FP32 settings in Appendix D.2, using two 100-step validation streams. Timing covers CPU input through CPU output, including refreshes and averaging, but excludes one-time graph preparation. Forecasts and recursive states match elementwise before and after optimization. When idle, the implementation changes reduce the latency of the $J = 2$ decoder by 60.6% for Timer, 41.9% for TimesFM.

E Ensemble and Hyperparameter Analyses

E.1 Refinement Budget and Ensemble Size

Table 9 gives the complete seven-dataset grid for $J \in \{1, 2\}$ and $M \in \{1, 2, 4, 8, 16, 32\}$ at $H = 720$. At the first inference step, only the current forecast is available; subsequently, each lead averages only historical forecasts that cover its target timestamp.

Longer histories lower MSE throughout the sweep on ETTh1, ETTm1, and Traffic. In contrast, intermediate sizes are preferable on ETTh2 and Electricity. ETTm2 and Weather worsen as history grows.

Furthermore, a second refinement round lowers MSE at every tested M on five datasets, most noticeably on ETTm1. On ETTm1, even the largest one-round ensemble has higher MSE than two rounds without averaging (0.578 versus 0.520). History therefore complements, rather than replaces, revision of the current forecast. The additional $J = 3$ sweep on ETTh1, ETTm1, ETTm2, and Electricity changes MSE by -0.10–$+0.24\%$ relative to $J = 2$, with small increases at every tested M on ETTm1 and ETTm2.

Figure 12 supplements Figure 4 with ETTh2, Weather, and Traffic for $J \in \{1, 2\}$.

Refresh period. We compare $R \in \{4, 16, 96\}$ on 721 consecutive ETTh1 validation origins using Timer with $H = 720$, $J = 2$, and $M = 8$. The MSEs are 1.0739, 1.0742, and 1.0736, respectively, differing by less than 0.06% relative to $R = 96$.

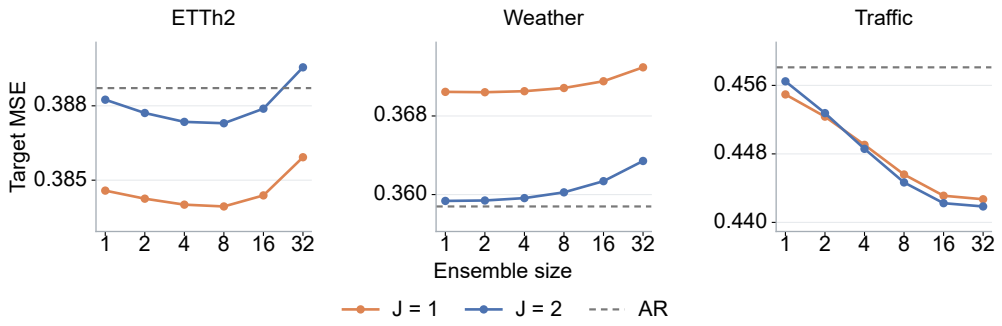

Figure 12: Timer MSE versus ensemble size on three additional datasets.

Table 9: Timer MSE versus ensemble size and refinement rounds at $H = 720$.

Dataset	$M = 1$	$M = 2$	$M = 4$	$M = 8$	$M = 16$	$M = 32$
$J = 1$						
ETTh1	0.434	0.433	0.431	0.429	0.427	0.427
ETTh2	0.385	0.384	0.384	0.384	0.384	0.386
ETTm1	0.687	0.676	0.661	0.640	0.612	0.578
ETTm2	0.398	0.398	0.398	0.398	0.399	0.400
Weather	0.370	0.370	0.371	0.371	0.372	0.373
Electricity	0.220	0.219	0.217	0.216	0.214	0.215
Traffic	0.455	0.452	0.449	0.446	0.443	0.443
$J = 2$						
ETTh1	0.433	0.431	0.429	0.427	0.425	0.425
ETTh2	0.388	0.387	0.387	0.387	0.387	0.389
ETTm1	0.520	0.518	0.516	0.512	0.506	0.497
ETTm2	0.397	0.397	0.397	0.397	0.398	0.400
Weather	0.359	0.359	0.360	0.360	0.361	0.363
Electricity	0.220	0.218	0.216	0.214	0.213	0.213
Traffic	0.456	0.453	0.449	0.445	0.442	0.442

E.2 When Does Averaging Improve Accuracy?

For a target timestamp, let $\mathbf{p}_0$ be the current prediction and $\mathbf{p}_0, \ldots, \mathbf{p}_{m-1}$ the m available aligned predictions. Writing $\bar{\mathbf{p}} = m^{-1} \sum_a \mathbf{p}_a$, the squared-error identity gives:

$$\|\bar{\mathbf{p}} - \mathbf{y}\|_2^2 - \|\mathbf{p}_0 - \mathbf{y}\|_2^2 = A - V, \tag{16}$$

$$A = \frac{1}{m} \sum_a \|\mathbf{p}_a - \mathbf{y}\|_2^2 - \|\mathbf{p}_0 - \mathbf{y}\|_2^2, \qquad V = \frac{1}{m} \sum_a \|\mathbf{p}_a - \bar{\mathbf{p}}\|_2^2.$$

The identity follows by expanding around $\bar{\mathbf{p}}$ and using $\sum_a (\mathbf{p}_a - \bar{\mathbf{p}}) = 0$. It requires neither independent nor unbiased errors. Averaging improves on the current forecast when the disagreement term V exceeds the additional average member loss A. This explains both the opportunity to combine complementary predictions and the cost of retaining less accurate historical members.

Table 10 evaluates this balance on the seven-dataset benchmark. The reported MSE terms average the coordinate-wise identity over channels, origins, and leads. Specifically, for both AR and $J = 2$, $V > A$ on ETTh1, ETTh2, ETTm1, Electricity, and Traffic; the order reverses on ETTm2 and Weather.

Table 10: Decomposition of averaging gains in 10^{-3} MSE units.

	AR			$J = 2$		
Dataset	A	V	$A - V$	A	V	$A - V$
ETTh1	+1.554	9.696	-8.141	+1.512	8.296	-6.783
ETTh2	+1.638	2.848	-1.210	+1.606	3.406	-1.800
ETTm1	+2.671	10.729	-8.058	+2.634	10.211	-7.577
ETTm2	+2.077	1.964	+0.112	+2.064	1.944	+0.120
Weather	+1.977	1.137	+0.840	+2.011	1.122	+0.889
Electricity	+1.156	8.491	-7.336	+1.138	7.225	-6.086
Traffic	+1.273	14.824	-13.552	+1.243	13.023	-11.780

E.3 Aggregation Rules

Table 11 compares current-only output, uniform means, medians, recency weights proportional to 2^{-a} for forecast age a, and a validation-selected rule (Selected). Each uses up to eight timestamp-aligned forecasts, excluding the first seven test origins. Selected chooses among fixed rules and nonnegative aggregation weights fitted on training data, with the forecasting model frozen. It yields only marginal MSE reductions over uniform averaging and sometimes performs worse. We therefore use uniform averaging, which achieves most of the gain with a simple, training-free rule that requires no dataset-specific weight fitting.

Table 11: MSE under fixed and validation-selected aggregation at $H = 720$.

	AR					$J = 2$				
Dataset	Current	Mean	Median	Recency	Selected	Current	Mean	Median	Recency	Selected
ETTh1	0.434	0.426	0.427	0.429	0.426	0.434	0.427	0.428	0.429	0.426
ETTh2	0.389	0.388	0.388	0.388	0.388	0.389	0.387	0.388	0.388	0.388
ETTm1	0.521	0.513	0.517	0.516	0.510	0.520	0.512	0.516	0.515	0.509
ETTm2	0.398	0.398	0.399	0.398	0.397	0.397	0.397	0.398	0.397	0.396
Weather	0.359	0.360	0.360	0.359	0.358	0.359	0.360	0.361	0.359	0.359
Electricity	0.221	0.213	0.214	0.216	0.213	0.220	0.214	0.214	0.216	0.213
Traffic	0.458	0.445	0.445	0.449	0.444	0.457	0.445	0.446	0.449	0.444

F Mechanisms of Temporal Reuse

F.1 Temporal Drafts Keep Predictions Closer to AR

Initializing with the previous forecast produces a much closer approximation to full AR decoding than initializing with the last observation or a linear extrapolation (Table 12). We compare these initializations with the same current context, exact first token, normalization, and two refinement rounds. Linear extrapolation is fitted by ordinary least squares to the last 96 observations. The settings are fixed using a 96-origin ETTh1 training probe, then evaluated on ETTh1, ETTh2, Weather, and ETTm2.

With a temporal draft, suffix divergence from AR is the smallest at every non-refresh step on all four datasets. The previous forecast already contains the model's predicted trajectory, so two refinement rounds can revise it using the new observations without constructing the whole suffix from a simple initial guess. This comparison measures agreement with the pretrained AR model. Forecasting error depends on both the magnitude and direction of the remaining deviation, as the decomposition in Appendix A.4 shows.

In Table 12, Div. is the mean squared difference from AR over forecast steps 289–720, computed before rolling averaging in current-window normalized coordinates. And ΔMSE the percentage change in target MSE relative to AR.

Table 12: Forecast accuracy and AR divergence with different draft initializations.

	ETTh1		ETTh2		Weather		ETTm2	
Initializer	Div.	ΔMSE	Div.	ΔMSE	Div.	ΔMSE	Div.	ΔMSE
Persistence	0.343	+11.9	0.349	-37.6	0.451	+50.4	0.349	+15.0
Linear	1.850	+109.3	0.875	-38.4	2.525	+145.2	1.825	+70.2
Temporal draft	**0.006**	-0.42	**0.010**	+0.24	**0.011**	-0.09	**0.011**	-0.21

In addition, refining this draft improves on directly copying it: with three model calls, the same 288-point AR prefix, and $R = 96$, the unaveraged $J = 2$ decoder lowers MSE on all eight datasets and MAE on seven compared with shift-and-paste (Table 2).

F.2 A One-Step Window Shift Does Not Preserve the Input KV Cache

Two consecutive Timer windows share 671 of 672 observations, yet moving the window by one time step changes all seven 96-point input patches. Each patch loses its first value and receives a value from the next patch; the last patch receives the new observation. The cached tokens therefore no longer represent the current patches. Below, we show where an exact update is possible, why it does not carry through the Transformer, and how the resulting cache errors affect predictions.

Linear patch embeddings admit an exact update. For one channel, let $\mathbf{x}_j^t \in \mathbb{R}^P$ be raw patch j, $\mathbf{A}$ the matrix that shifts its coordinates left and fills the last coordinate with zero, and $\mathbf{b}$ the last coordinate vector. If c_j^t is the incoming boundary value, then:

$$\mathbf{x}_j^{t+1} = \mathbf{A}\mathbf{x}_j^t + \mathbf{b}c_j^t. \tag{17}$$

Let μ_t and $\sigma_t > 0$ denote the window mean and scale, and write the linear embedding as $\mathbf{e}_j^t = \mathbf{W}(\mathbf{x}_j^t - \mu_t\mathbf{1})/\sigma_t$, with any fixed embedding bias subtracted. When $\mathbf{W}$ has full column rank, its left inverse satisfies $\mathbf{W}^\dagger\mathbf{W} = \mathbf{I}_P$. Substituting Equation 17 and using $\mathbf{A}\mathbf{1} = \mathbf{1} - \mathbf{b}$ gives:

$$\mathbf{e}_j^{t+1} = \frac{\sigma_t}{\sigma_{t+1}}\mathbf{W}\mathbf{A}\mathbf{W}^\dagger\mathbf{e}_j^t + \frac{\mu_t - \mu_{t+1}}{\sigma_{t+1}}\mathbf{W}\mathbf{1} + \frac{c_j^t - \mu_t}{\sigma_{t+1}}\mathbf{W}\mathbf{b}. \tag{18}$$

Thus the old embedding can be updated exactly using a linear map, the new window statistics, and a boundary correction. Timer's 1024×96 embedding matrix has rank 96 and satisfies this condition. Moving one raw time step requires a change within every token, rather than a shift of token indices.

Attention does not inherit this linear update. Linear projections preserve exact additive updates. For example, if a hidden state changes by $\Delta\mathbf{H}$, its key projection changes by $\Delta\mathbf{H}\mathbf{W}_K$. The difficulty is obtaining that exact hidden-state update after attention and the nonlinear layers. For a single attention head, write:

$$\mathbf{S} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}_{\mathrm{causal}}\right), \qquad \mathbf{O} = \mathbf{S}\mathbf{V}, \tag{19}$$

where the softmax acts row-wise and $\mathbf{M}_{\mathrm{causal}}$ is the causal mask. With primes denoting the next window, the exact output change is:

$$\mathbf{O}' - \mathbf{O} = (\mathbf{S}' - \mathbf{S})\mathbf{V} + \mathbf{S}'(\mathbf{V}' - \mathbf{V}). \tag{20}$$

The first term depends on the new queries and keys through their pairwise scores and softmax. The second depends on the new values. A fixed linear update of the old output does not generally recover both terms. Moreover, layer normalization and nonlinear feed-forward activations depend on the updated state, so Equation 18 cannot simply be applied to every layer's KV cache. Reassigning positions alone cannot correct these changes in token content and context.

The layerwise experiments show partial recovery. We fit a separate orthogonal Procrustes map T_ℓ at each hidden-state depth using 1,345 origins from the first half of ETTh1 validation, then freeze it for the other half (1,345 origins) and ETTh2 test (2,785 origins). For paired states $\mathbf{H}_\ell^t, \mathbf{H}_\ell^{t+1}$, the fraction of the stale-state error removed is:

$$R_\ell = 1 - \frac{\sum_t \|\mathbf{H}_\ell^{t+1} - T_\ell(\mathbf{H}_\ell^t)\|_F^2}{\sum_t \|\mathbf{H}_\ell^{t+1} - \mathbf{H}_\ell^t\|_F^2}. \tag{21}$$

Here, zero corresponds to retaining the old state and one to an exact update. Figure 13(a) shows that the fitted maps remove about 97% of the error at the embedding layer, 20–23% at depth six, and 58–75% at the final layer. The reusable linear structure therefore varies substantially with depth and does not provide an exact update for the entire cache.

Stale values account for most of the prediction deviation. We isolate the effects of K and V on all 2,785 ETTh1 test origins with $C = 672$, $H = 96$, and stride one. Starting from the exact preceding-window cache, we replace selected components at the first six token positions with their exact current-window counterparts and recompute the final input token. Table 13 reports the resulting prediction MSE relative to full recomputation and the percentage removed from the stale-KV discrepancy. Updating all keys removes only 2.6% of that discrepancy. In contrast, updating all values removes 97.1%. Updating both recovers full recomputation within 3.34×10^{-6} maximum absolute error. This agrees with Equation 20: keys affect the attention weights, and values directly supply the content combined by those weights.

Table 13: Prediction deviation after replacing cached keys or values on ETTh1.

Components updated	MSE vs. full recomputation	Discrepancy removed (%)
None (stale KV)	0.023454	0.0
K, final layer	0.023184	1.2
V, final layer	0.013705	41.6
K, final four layers	0.022921	2.3
V, final four layers	0.012003	48.8
K, all layers	0.022834	2.6
V, all layers	0.000686	97.1
K and V, all layers	6.29×10^{-14}	100.0

Approximation error grows over consecutive cache reuse. On the same ETTh1 test stream, we next carry the approximate cache forward recursively and refresh it every four steps. Between refreshes, keys are retained, values in the last four layers are interpolated between adjacent cached positions with weight 0.6875 on the following position, and the final input token is recomputed. Figure 13(b) shows that excess target MSE grows from 0.006691 to 0.021215 and 0.041368 over three consecutive reuse steps, then returns to zero on refresh. These four phases contain 697, 696, 696, and 696 origins, respectively. Full-stream MSE increases from 0.362955 to 0.380270 under training-split scaling. The growing error explains our choice to rebuild current-context KV at every inference step and reuse the previous forecast as a draft that is revised under that exact context.

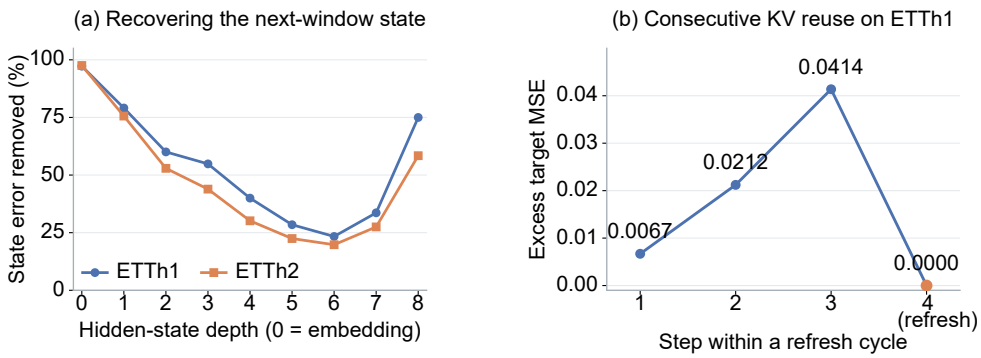

Figure 13: Layerwise state recovery and forecasting error during consecutive KV reuse.

G Qualitative Examples and Discussion

G.1 Historical Forecast Errors

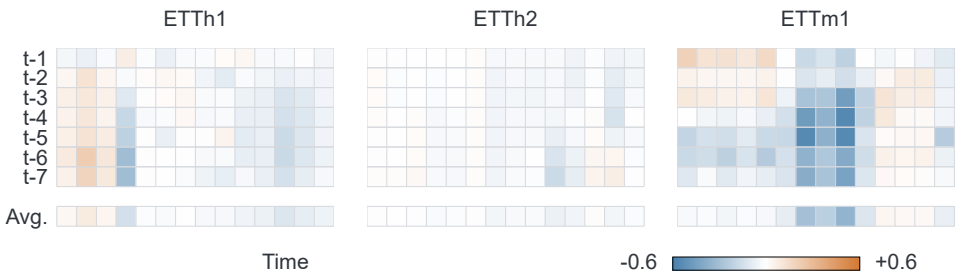

Figure 14: Error change from the current forecast on ETTh1, ETTh2, and ETTm1. Blue indicates improvement and orange indicates degradation.

G.2 Forecast Examples

Figures 15–17 show low-error examples in which TempoDraft has lower MSE than AR across five datasets and four backbones. Each panel reports MSE on the displayed 720-step interval in training-standardized units. Specifically, the examples show smaller amplitude errors on ETTm1 and Electricity and less downward drift on ETTm2.

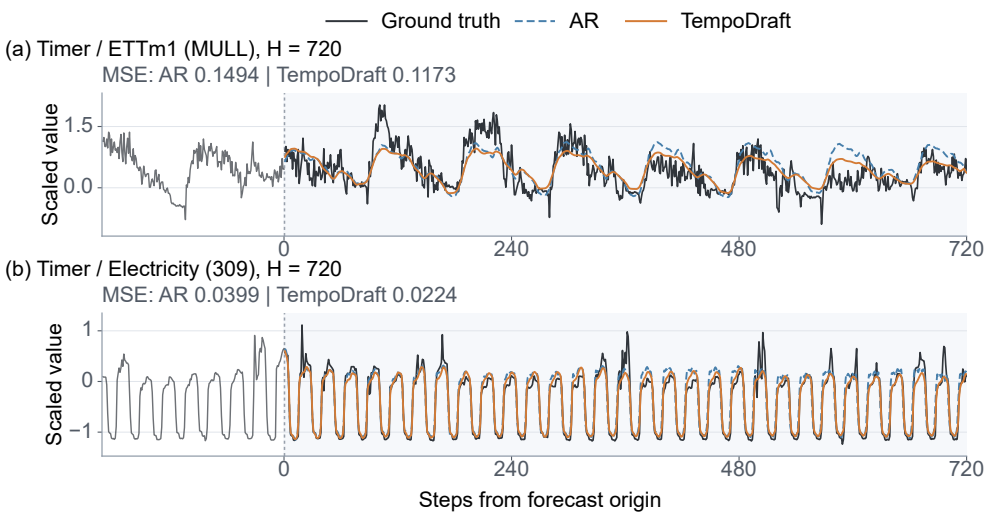

Figure 15: Example forecasts on ETTm1 and Electricity.

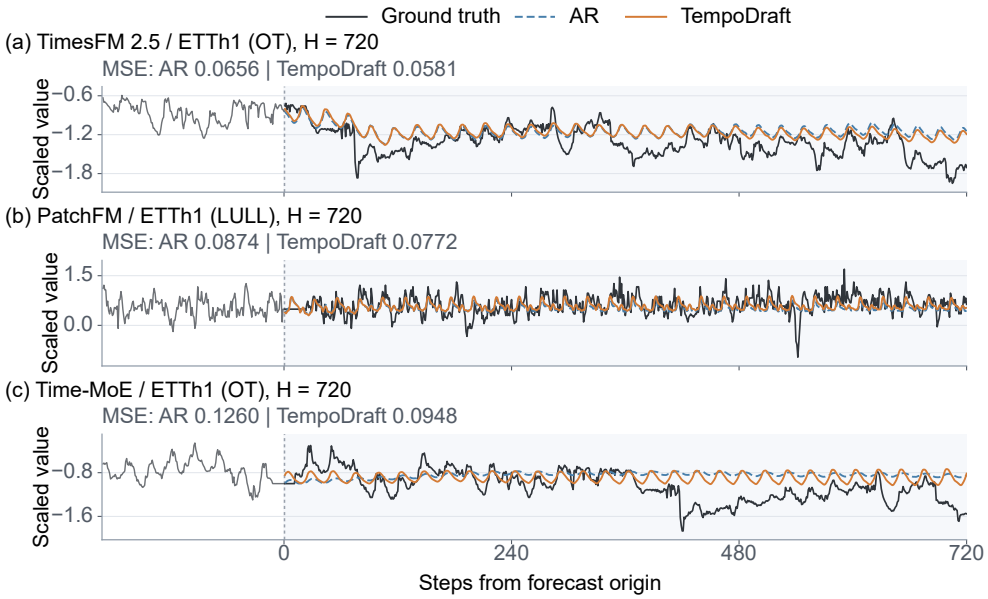

Figure 16: Forecasts from TimesFM, PatchFM, and Time-MoE on ETTh1.

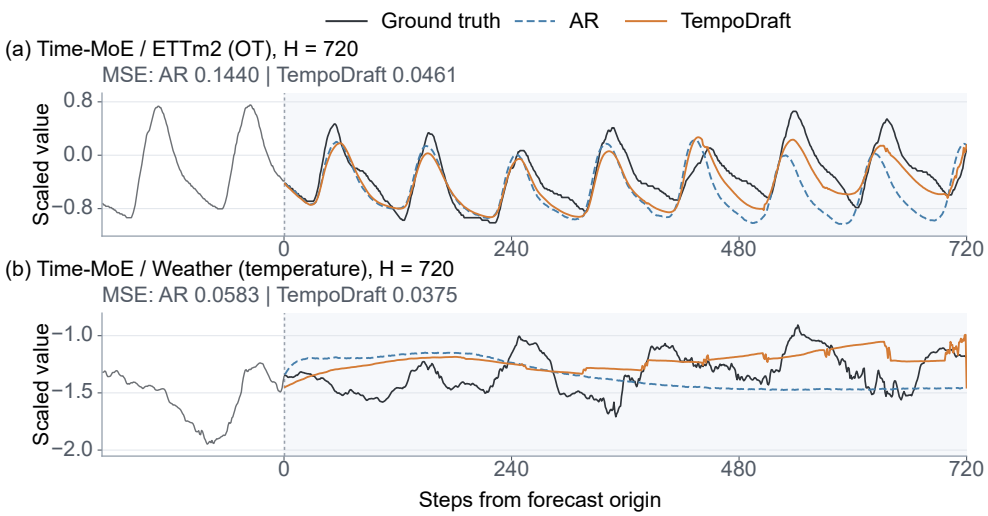

Figure 17: Time-MoE forecasts on ETTm2 and Weather.

G.3 Future Directions

Adaptive reuse. Future work could adjust the refinement budget and refresh interval using changes in the observed dynamics and forecast uncertainty. Such a policy would allocate more computation when a new observation calls for a larger revision.

Selecting forecast history. A theoretical analysis of how error correlations vary with temporal dependence and forecast age could guide the selection and weighting of past forecasts. This would extend fixed averaging to aggregation that adapts as the stream changes.

Learning to revise. Training on consecutive forecasting tasks could teach models to use earlier predictions explicitly. Jointly designing the forecasting objective and revision procedure may reduce the computation needed to update a forecast.