COPYTESTBENCH: CROSS-CHANNEL RETRIEVAL IN TRANSFORMER FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world forecasting benchmarks can hide important differences between models: similar errors may result from different computations, and failures are hard to interpret when the optimal predictor and irreducible error are unknown. We introduce CopyTestBench (CTBench), a diagnostic, capability-level benchmark with known rules and zero Bayes error, where the correct target always appears in the input window. Rather than only ranking forecasters by aggregate error, CTBench tests whether they retrieve, route, and preserve delayed cross-channel dependencies under *pure-copy*, *daily switching*, and *rotating switching*, with varying distractors and autocorrelation. Ridge regression nearly recovers the exact solution on pure-copy tasks and achieves the lowest overall error, showing that the dependency is observable and linearly solvable. Under the same training protocol, none of the transformer-based forecasters matches this accuracy. Several instead rely on the target level or switching pattern. Some models beat gate-aware persistence under daily switching, but none do so under rotating switching. Controlled experiments identify failures in source routing and normalization. Vanilla Transformer solves one pure-copy dataset under a constant learning rate, while iTransformer remains near the window-mean baseline. Oracle routing, combined with changes to per-channel instance normalization and encoder layer normalization, raises R^2 from 0.101 to 0.987, with consistent results across seeds and on an exact-copy task built from real measurements. CTBench therefore connects controlled capability evaluation to mechanistic diagnosis, providing a way to uncover weaknesses hidden by real-world benchmarks and guide the development of stronger Transformer-based forecasters.

1 INTRODUCTION

Transformer forecasters differ in how they tokenize a series, mix information across channels, normalize inputs, and train (Zhou et al., 2021; Wu et al., 2021; Zhou et al., 2022; Nie et al., 2023; Liu et al., 2024; Wang et al., 2024a; Ilbert et al., 2024). Yet linear models can match or outperform them on standard benchmarks (Zeng et al., 2023). A useful question is therefore not only which model has the lowest error, but also *what computation produced that error*.

Real-world forecasting data rarely answer that question. The correct forecast and the irreducible error are unknown. A poor score may reflect a missed dependency or an unpredictable residual; a good score may reflect the intended dependency or a simple persistence-like shortcut. This distinction matters especially for cross-channel retrieval, which may add little on data dominated by a target's own history. Prior work has highlighted the competitiveness of simple models, the role of normalization, and the limits of current benchmark practice (Zeng et al., 2023; Chen et al., 2025; Qiu et al., 2024; Zhou et al., 2025). These benchmark critiques can reveal questionable rankings, but they generally cannot determine whether a forecaster recovered a specified cross-channel computation or identify which architectural operation prevented it. Our goal is to connect a controlled capability test to a mechanistic diagnosis and intervention.

We introduce *CopyTestBench* (CTBench), diagnostic tasks for which the rule, the optimal forecast, and the attainable error are known in advance. In the main family, *Gated Mixture*, the target is a delayed copy of one of two observed random-walk sources, and a gate selects which one: the gate is either constant (*pure-copy*) or follows a calendar-based switching rule (*daily switching* and

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.

rotating switching). Lookback, horizon, and delay all equal 96 steps, so the value to be copied is always inside the input window. Crossing three selection rules with three distractor counts and three distractor autocorrelations gives 27 configurations. A second family, *Random Causal*, replaces the random walks with a sparse pulse train and removes calendar structure. CTBench is thus a capability-level unit test, not a general ranking of forecasters: it asks whether a model can retrieve, route, and preserve an observed source signal.

Under one shared training protocol, ridge regression solves the pure copy essentially exactly, whereas the best neural model reaches $R^2 = 0.738$ and iTransformer stays near level-based baselines. This is not simply a tuning artifact: a learning-rate sweep lets Vanilla Transformer solve the task on the one dataset where it was run ($R^2 = 0.967$) but never rescues iTransformer (Appendix E.3). The iTransformer failure persists on five independently generated pairs of source paths (a source path, also called *source draw*, is one independently generated random walk; we generate pairs of them and use them as input to the data generators). The picture changes with the source process: with stationary AR(1) sources, Vanilla Transformer largely recovers the copy while iTransformer does not, and on Random Causal sources iTransformer reaches $R^2 = 0.907$ without a distractor but only 0.201 with one distractor. Our results do not show that transformers are intrinsically unable to copy, but they interestingly highlight that *retrieval depends on the source process, the distractors, and the architecture.*

The known rule also lets us diagnose one iTransformer failure in detail. Attention weights, input-output Jacobians and layer-wise probes point to three interacting bottlenecks: weak source routing, loss of cross-channel offsets under per-channel instance normalization, and amplitude rescaling by encoder layer normalization. We test this diagnosis by intervention: forcing the target token to attend to the correct source and changing both normalization stages raises R^2 from 0.101 to 0.987. Because the route is supplied rather than learned, this is a mechanistic test, not a deployable method, which shows that *the architecture can express the solution and localize where training fails.* On a copy task built from real ETTh1 measurements, both architectures reach substantial positive R^2 under standard training, and the same intervention reaches $R^2 = 0.978$, so *the intervention also works on this real-measurement copy task.*

Our contributions are: *(i)* CTBench, exact-copy forecasting tasks with zero Bayes error and simple reference baselines that permit to diagnose retrieval capabilities of transformer-based forecasters; *(ii)* an evaluation of twelve forecasters (ten transformers, deterministic ridge, and XGBoost), plus five independent source draws for eight of them, namely iTransformer, Vanilla Transformer, and six channel-wise models; *(iii)* a controlled case study that localizes iTransformer's failure to source routing and two normalization stages, showing the capability of the benchmark to diagnose transformer forecaster weaknesses and related potential for methodological improvement; *(iv)* design and reporting guidelines for diagnostic forecasting tasks. Beyond these results, CTBench raises three open questions for the community, that could guide the development of stronger Transformer-based forecasters: *Q1.* Why do some transformer forecasters fail an exact, fully observable copy, and which architectural changes make them robust to it? *Q2.* Can copy-test error be attributed to specific architectural components, and does repairing them reduce the error? *Q3.* To what extent do gains on the copy test translate into better forecasts on real-world data?

2 Related Work

Transformer forecasters and their benchmarks. Early methods use one token per time step (Zhou et al., 2021; Wu et al., 2021; Zhou et al., 2022). PatchTST uses patches and forecasts each channel independently (Nie et al., 2023); iTransformer uses one token per variable and applies attention across channels (Liu et al., 2024); Crossformer and CARD combine temporal and cross-channel attention (Zhang & Yan, 2023; Wang et al., 2024a). SAMformer addresses optimization through a shallow channel-wise encoder, sharpness-aware minimization (Foret et al., 2021), and RevIN (Kim et al., 2022). These attention strategies and components are usually evaluated together on a small collection of real datasets, making their individual contributions difficult to separate. Linear baselines can be competitive (Zeng et al., 2023), and mutual-information analyses emphasize normalization and intra-variate structure over inter-variate attention (Chen et al., 2025). We do not propose a new architecture, but a diagnostic benchmark that tests a specific capability of existing ones, which standard real-world benchmarks cannot isolate.

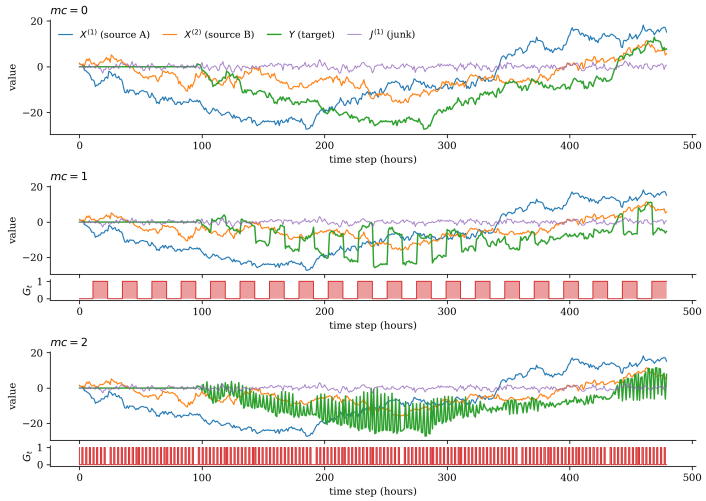


Figure 1: Datasets from the three Gated Mixture source selection configurations (distractor count $= 1$, $\rho = 0.5$). Each panel overlays the two sources, the target and one distractor channel, and the red strip below the lower two panels shows the gate G_t. At mc=0 the target is a lagged copy of source $X^{(1)}$. At mc=1 the gate switches at noon and midnight. At mc=2 the gate selects the second source every three hours with a phase that rotates daily, so the pattern is not periodic in 24 hours.

Synthetic and theoretical diagnostics. Ilbert et al. (2024) study a simple linear forecasting problem whose solution a transformer can represent but does not learn, a close precedent for our construction. Zhou et al. (2025) establish limits of linear self-attention on autoregressive data and show that recursively generated predictions decay toward the mean. Two concurrent studies are also relevant. Garcia & Remy (2026) compare forecasters with optimal predictors of synthetic processes in low signal-to-noise settings. Their two-way attention transformer outperforms Lasso and boosting, and their attention analysis is observational. Gonon et al. (2026) analyze optimization of a match-and-copy mechanism in a minimal language-model retrieval task. Our original contribution is represented by the combination of a zero-Bayes-error forecasting benchmark with published forecasters, simple reference baselines, and internal interventions (Table 3 in Appendix A summarizes how this differs from prior work).

Normalization and shortcut learning. RevIN (Kim et al., 2022) introduced reversible instance normalization for forecasting. Subsequent work noted that normalization can remove useful level information (Liu et al., 2022; Fan et al., 2023; Liu et al., 2023). The known optimum in CTBench allows it to distinguish the recovery of required dependency from reliance on a simpler predictive feature, as studied in shortcut learning (Geirhos et al., 2020; Shah et al., 2020). Whereas language models can copy through induction heads (Olsson et al., 2022), our analysis examines why related retrieval behavior is not reliably learned in the evaluated forecasting setting. Appendix A provides further comparisons.

3 CTBench: A Diagnostic Benchmark with Zero Bayes Error

3.1 Dataset Generators

CTBench has two generator families. A configuration sets the generator parameters, and a dataset is one time-series realization generated from those settings and a random seed. We observe N channels hourly, writing $\mathbf{Z}_t \in \mathbb{R}^N$ for the observation, $\tau(t)$ for its timestamp and Y_t for the target. A forecaster observes all channels over L steps and predicts the next H target values. We set the source-to-target delay to ℓ and use $L = H = \ell = 96$. Each dataset has $T = 5{,}000$ steps, split $70/10/20$ into training, validation and test sets, yielding 3,309 training and 905 test windows[1].

[1]The Python code with generator models and the related datasets will be made public upon acceptance.

Gated Mixture (primary family). Two independent Gaussian random walks, initialized at zero and with unit-variance increments, provide the sources. A deterministic binary source selection variable, or *gate*, $G_t = g_m(\tau(t))$, chooses which delayed source defines the target:

$$Y_t = (1 - G_t)\, X^{(1)}_{t-\ell} + G_t\, X^{(2)}_{t-\ell}, \qquad t > \ell, \tag{1}$$

with $Y_t = 0$ during the initial period $t \le \ell$. Let h_t and d_t denote hour of day and day of week. The source selection parameter mc specifies three generator variants, called respectively mc=0, mc=1, and mc=2 in the following:

$$g_0 \equiv 0, \qquad g_1 = \mathbf{1}\big[h_t \ge 12\big], \qquad g_2 = \mathbf{1}\big[(h_t + d_t) \bmod 3 = 0\big], \tag{2}$$

These correspond to *pure-copy*, *daily switching*, and *rotating switching* (see Figure 1). We add N_J irrelevant, or *distractor*, channels with no causal connection to the target. Each follows $J^{(n)}_t = \rho\, J^{(n)}_{t-1} + \eta^{(n)}_t, n = 1, \ldots, N_J$, where the innovation variance is set so that the process has unit stationary variance. Thus $\rho = 0$ gives white noise and $\rho = 0.9$ gives strong temporal dependence. The observed vector of independent and dependent variables in each dataset is therefore $\mathbf{Z}_t = (X^{(1)}_t, X^{(2)}_t, G_t, J^{(1)}_t, \ldots, J^{(N_J)}_t, Y_t)$, where Y_t is the dependent variable. Namely, both the gate G_t and target history Y_t are observed. The models also receive calendar features derived from the timestamp $\tau(t)$, including the hour of the day h_t and day of week d_t timestamps use the standard hourly calendar encoding. The factorial design combines $mc \in \{0, 1, 2\}$, $N_J \in \{1, 3, 5\}$ and $\rho \in \{0, 0.5, 0.9\}$. It yields 27 configurations and 27 main-grid datasets, nine for each source-selection rule; they share one pair of source paths. No noise is added to Y.

Random Causal (secondary family). This generator removes calendar structure and replaces drifting sources with a sparse pulse train. The target is a delayed copy of that source, so accurate prediction requires cross-channel retrieval. We evaluate the generator with and without a distractor. Its specification and results are in Appendix I.

3.2 Zero Bayes error and standardized evaluation

Because $L = H = \ell$, the source value needed for forecast step $h = 1, \ldots, H$ is at step $t + h - \ell$, within the observed window. Thus $Y_{t+h} = f_h(\mathbf{Z}_{t-L+1:t}, \tau)$ exactly, where f_h selects the source using g_m and copies its value at that index. Because the required source values and switching rule are fully known from the input, the target is exactly predictable. Any remaining error therefore comes from the forecasting model and its training pipeline. All pipelines standardize each channel using its training-set mean μ^{tr}_c and standard deviation σ^{tr}_c; we write $\tilde{X}$ and $\tilde{Y}$ for the standardized values. For $t + h > \ell$, let $1 + G_{t+h}$ denote the selected source. Equation 1 then becomes

$$\widetilde{Y}_{t+h} = (1 - G_{t+h}) \left(\frac{\sigma^{\mathrm{tr}}_{X^{(1)}}}{\sigma^{\mathrm{tr}}_Y}\, \widetilde{X}^{(1)}_{t+h-\ell} + \frac{\mu^{\mathrm{tr}}_{X^{(1)}} - \mu^{\mathrm{tr}}_Y}{\sigma^{\mathrm{tr}}_Y} \right) + G_{t+h} \left(\frac{\sigma^{\mathrm{tr}}_{X^{(2)}}}{\sigma^{\mathrm{tr}}_Y}\, \widetilde{X}^{(2)}_{t+h-\ell} + \frac{\mu^{\mathrm{tr}}_{X^{(2)}} - \mu^{\mathrm{tr}}_Y}{\sigma^{\mathrm{tr}}_Y} \right). \tag{3}$$

Thus training-set standardization turns the raw delayed copy into an exact affine copy. Appendix C gives the approximate coefficients for these datasets. Its nonzero offset matters for the normalization analysis in Section 5.

3.3 Reference baselines, R^2 and success criteria

We report test MSE on the standardized target alongside three simple reference baselines. *Persistence* (PE) repeats the last observed target value; *window mean* (WM) repeats the lookback mean. *Gate-aware persistence* (GP) uses the true switching schedule and, for each forecast step, repeats the most recent lookback target value with the same gate state. GP tests whether a model improves on schedule knowledge without detailed source retrieval. We also report $R^2 = 1 - \mathrm{MSE}/\mathrm{Var}_{\text{test}}(Y)$ using the variance of the evaluated target windows. At mc=0, this variance is 0.046, so MSE near 0.04 may reflect little more than level tracking, which is predicting the target's overall level without recovering its within window variation. We use two success levels. A model *solves* a dataset if $R^2 \ge 0.95$. It shows *partial recovery* on a source selection configuration (mc=0, mc=1 or mc=2) if its MSE is below that of the best simple reference baseline on all nine datasets of that configuration; otherwise it fails the criterion, even if its raw MSE is small. Raw R^2 still credits a model for predicting each window's level. To compare source draws, whose test blocks differ strongly in

variance, Appendix E.1 therefore also reports a *shape* R^2, which removes each window's mean from both forecast and target $R^2_{\text{shape}} = 1 - \frac{\sum_{i,h}\left[(\hat{y}_{ih}-\bar{\hat{y}}_i)-(y_{ih}-\bar{y}_i)\right]^2}{\sum_{i,h}(y_{ih}-\bar{y}_i)^2}$.

3.4 Limits of the switching configurations and a design principle

With daily-switching source selection (mc=1), the switching pattern repeats every 24 hours. A 96-step lookback contains four cycles, allowing the future schedule to be inferred without calendar features. Section 5 confirms this through retraining. The rotating-switching configuration (mc=2) has a related limitation. Because d_t resets weekly, its true period is 168 hours, not a clean 3-day rotation; this period does not divide 96, but any four-day window still contains all three daily phases and their one-step progression. Removing calendar features again has limited effect. We therefore interpret both configurations as tests of source switching, not isolated tests of calendar use. This suggests two necessary design conditions: *the switching period should not divide the lookback or horizon, and the schedule observed in the lookback should not determine the subsequent schedule*. Neither current switching configuration fully meets both conditions.

4 Experimental Setup

Evaluated models. Following Wang et al. (2024b), we distinguish point-wise tokens (one time step across channels), patch-wise tokens (a block of steps within a channel) and channel-wise tokens (one channel's complete window). Channel-wise attention provides explicit cross-channel weights for inspection and intervention. **Vanilla Transformer** (VanTF) (Vaswani et al., 2017) is our point-wise encoder-decoder reference; it does not reproduce all components of Informer, Autoformer or FEDformer. **iTransformer** (Liu et al., 2024) uses channel-wise tokens and input/output instance normalization. The SAMformer family (Ilbert et al., 2024) comprises **Simple TF**, a shallow transformer with channel-wise tokens whose self-attention mixes information across channels (as in iTransformer), and its variants with **RevIN**, **SAM** or both (**SAMformer**), and a **spectral-norm** variant with and without RevIN. **CARD** (Wang et al., 2024a) combines temporal and channel attention. **PatchTST** (Nie et al., 2023) forecasts channels independently. Classical alternatives are **ridge regression** on the flattened multivariate window and multivariate **XGBoost** (Chen & Guestrin, 2016). The three simple reference baselines complete the comparison (Table 4, Appendix D).

Protocol. Deep models use $d_{\text{model}} = 512$, 8 heads, 2 encoder layers, $d_{\text{ff}} = 2048$, dropout 0.1, learning rate 10^{-3} and batch size 32. Training lasts 20 epochs without early stopping; testing uses the best-validation checkpoint. PatchTST instead uses its released 3-layer patch configuration. The shared settings are within iTransformer's released range (learning rate 5×10^{-4} to 10^{-3}, width 512, two layers, 10 epochs). SAMformer's released defaults differ substantially (lookback 336, 100 epochs, learning rate 10^{-4}), yet its family includes the best neural results here. These comparisons provide protocol context, not evidence of optimal tuning. Each attention model is averaged over three seeds. Ridge is deterministic, with validation-selected regularization (Appendix E.4).

Statistical reporting. We train each neural model three times per dataset, changing the training seed but keeping the data fixed. Each run produces one test MSE, and we report the average of the three. Appendix E summarizes how much these scores change when a model is retrained on the same data.

5 Results and discussions

Results across datasets. Table 1 reports all 27 configurations with per-rule and overall means (seed variation in Appendix E). Ridge has the lowest MSE on all 18 configurations with $mc \leq 1$ and the lowest overall mean, 0.037. Among the remaining learned forecasters, the overall ordering is Simple TF + RevIN (0.043), SAMformer (0.051), Spectral TF + RevIN (0.052), XGBoost (0.054), and iTransformer (0.055). At mc=0 some transformer-based models (green cells) beat the best simple reference baselines (PE and GP, MSE 0.040); at mc=1 several learned models beat the best simple reference baseline (GP with test MSE 0.070), and ridge more than halves GP's error; at mc=2, the GP baseline reference reaches a test MSE of 0.055 and beats every learned model, including ridge and XGBoost, on all nine configurations.

	mc	N_J	ρ	iTrans	VanTF	SmpTF	+RevIN	+SAM	SAMf	SpTF	SpTF+R	CARD	PTST	Ridge	XGB	PE	WM	GP
Pure-copy	0	1	0.0	0.046	0.029	0.120	0.023	0.042	0.017	0.101	0.014	0.016	0.047	**0.000**	0.011	0.040	0.047	0.040
	0	1	0.5	0.041	0.037	0.169	0.021	0.042	0.018	0.106	0.014	0.017	0.047	**0.000**	0.011	0.040	0.047	0.040
	0	1	0.9	0.047	0.024	0.356	0.022	0.042	0.019	0.100	0.014	0.024	0.047	**0.000**	0.011	0.040	0.047	0.040
	0	3	0.0	0.045	0.038	0.122	0.021	0.041	0.018	0.103	0.012	0.021	0.047	**0.000**	0.010	0.040	0.047	0.040
	0	3	0.5	0.049	0.034	0.102	0.022	0.042	0.016	0.109	0.014	0.019	0.047	**0.000**	0.011	0.040	0.047	0.040
	0	3	0.9	0.045	0.030	0.270	0.025	0.043	0.022	0.056	0.014	0.021	0.045	**0.000**	0.011	0.040	0.047	0.040
	0	5	0.0	0.045	0.044	0.088	0.021	0.041	0.023	0.109	0.014	0.020	0.048	**0.000**	0.011	0.040	0.047	0.040
	0	5	0.5	0.042	0.063	0.083	0.024	0.042	0.022	0.100	0.012	0.019	0.045	**0.000**	0.011	0.040	0.047	0.040
	0	5	0.9	0.045	0.052	0.151	0.025	0.045	0.020	0.068	0.013	0.025	0.049	**0.000**	0.011	0.040	0.047	0.040
	mean		mc=0	0.045	0.039	0.162	0.023	0.042	0.019	0.095	0.013	0.020	0.047	**0.000**	0.011	0.040	0.047	0.040
Daily switching	1	1	0.0	0.051	0.041	0.105	0.040	0.066	0.057	0.157	0.054	0.047	0.082	**0.032**	0.058	0.576	0.337	0.070
	1	1	0.5	0.052	0.037	0.106	0.043	0.076	0.053	0.195	0.054	0.044	0.080	**0.032**	0.057	0.576	0.337	0.070
	1	1	0.9	0.058	0.041	0.121	0.041	0.079	0.056	0.354	0.056	0.048	0.080	**0.032**	0.061	0.576	0.337	0.070
	1	3	0.0	0.051	0.043	0.074	0.042	0.069	0.055	0.180	0.058	0.050	0.084	**0.030**	0.062	0.576	0.337	0.070
	1	3	0.5	0.051	0.043	0.072	0.042	0.079	0.053	0.167	0.056	0.058	0.083	**0.030**	0.063	0.576	0.337	0.070
	1	3	0.9	0.059	0.051	0.120	0.042	0.082	0.065	0.289	0.056	0.058	0.082	**0.030**	0.063	0.576	0.337	0.070
	1	5	0.0	0.051	0.071	0.072	0.041	0.072	0.053	0.155	0.055	0.053	0.082	**0.032**	0.062	0.576	0.337	0.070
	1	5	0.5	0.054	0.059	0.084	0.043	0.080	0.056	0.146	0.055	0.053	0.081	**0.032**	0.061	0.576	0.337	0.070
	1	5	0.9	0.059	0.073	0.108	0.044	0.082	0.056	0.210	0.055	0.061	0.079	**0.032**	0.065	0.576	0.337	0.070
	mean		mc=1	0.054	0.051	0.096	0.042	0.076	0.056	0.206	0.056	0.052	0.081	**0.031**	0.061	0.576	0.337	0.070
Rotating switching	2	1	0.0	0.063	0.242	0.194	0.059	0.116	0.077	0.178	0.081	0.094	0.100	0.073	0.088	0.452	0.263	**0.055**
	2	1	0.5	0.062	0.242	0.222	0.061	0.119	0.077	0.194	0.081	0.093	0.104	0.073	0.090	0.452	0.263	**0.055**
	2	1	0.9	0.069	0.244	0.225	0.065	0.116	0.077	0.495	0.080	0.094	0.095	0.073	0.088	0.452	0.263	**0.055**
	2	3	0.0	0.060	0.251	0.159	0.063	0.121	0.078	0.186	0.088	0.109	0.101	0.079	0.090	0.452	0.263	**0.055**
	2	3	0.5	0.064	0.261	0.127	0.068	0.122	0.078	0.178	0.087	0.107	0.097	0.079	0.089	0.452	0.263	**0.055**
	2	3	0.9	0.071	0.252	0.251	0.068	0.119	0.081	0.385	0.090	0.112	0.095	0.079	0.088	0.452	0.263	**0.055**
	2	5	0.0	0.065	0.256	0.124	0.069	0.122	0.081	0.181	0.092	0.107	0.105	0.094	0.092	0.452	0.263	**0.055**
	2	5	0.5	0.068	0.258	0.126	0.069	0.124	0.080	0.180	0.090	0.105	0.093	0.094	0.093	0.452	0.263	**0.055**
	2	5	0.9	0.079	0.263	0.231	0.070	0.120	0.080	0.304	0.094	0.109	0.091	0.080	0.092	0.452	0.263	**0.055**
	mean		mc=2	0.067	0.252	0.184	0.066	0.120	0.079	0.253	0.087	0.103	0.098	0.080	0.090	0.452	0.263	**0.055**
	grand mean			0.055	0.114	0.148	0.043	0.079	0.051	0.185	0.052	0.059	0.075	**0.037**	0.054	0.356	0.216	0.055

Table 1: Test MSE on all 27 configurations (9 configurations for each of the 3 source-selection settings) of the Gated Mixture grid. Lower is better. The vertical labels identify the three source-selection settings. Every attention column is a three-seed mean, Ridge is deterministic (Appendix E.4), XGB is a single canonical run, and PE, WM and GP are the simple reference baselines defined in Section 3.3. **Bold** marks the row minimum over all fifteen columns, simple reference baselines included. The *mean* row after each block averages that source-selection type (e.g., mc=0) over nine cells and the last row averages all 27 configurations, with the row minimum in bold. SmpTF is Simple TF, +RevIN and +SAM are Simple TF with that addition, SAMf is SAMformer, SpTF is Spectral TF, SpTF+R is Spectral TF + RevIN, PTST is PatchTST (896 test windows, Appendix E.5). The same regime means with their across-seed spread are in Table 5.

1. Can transformers recover an exact, fully observable cross-channel dependency? (pure-copy, mc=0**)** At mc=0, the target is a delayed copy of one source, with no switching schedule to infer. Ridge solves every dataset with $R^2 \geq 0.999$, confirming that the rule is observable and linear under the shared preprocessing. XGBoost reaches $R^2 = 0.761$–0.776. iTransformer's MSE ranges from 0.041 to 0.049, close to persistence (PE/GP, 0.040) and window mean (WM, 0.047); its best R^2 is 0.103 (Figure 2, top row). VanTF reaches MSE 0.024–0.063 and R^2 up to 0.471 on its best dataset but performs worse than PE on all three N_J=5 datasets. The three RevIN-equipped channel-wise variants show partial recovery, with MSE 0.012–0.025 and R^2 up to 0.738, as do CARD (0.016–0.025) and XGBoost (0.011). PatchTST cannot access the source and remains near the window-mean baseline at 0.047–0.049. Thus *ridge solves the copy; the other positive controls improve on simple reference baselines without solving it; and iTransformer, VanTF, PatchTST, and the three non-RevIN variants fail the stated criterion.*

The random-walk source process makes level tracking an attractive shortcut. This cause the models to lower their MSE by fluctuating around the mean of each window. In the mc=0 training windows, differences in window means account for 98% of target variance, while within-window shape accounts for 2% (Appendix C). A model can therefore lower training loss by predicting the target's general level without identifying the required source values. This does not show that these architectures cannot copy; it shows that random-walk sources make this level shortcut nearly as good as the intended solution. This motivates the open **Q1** (see Section 1).

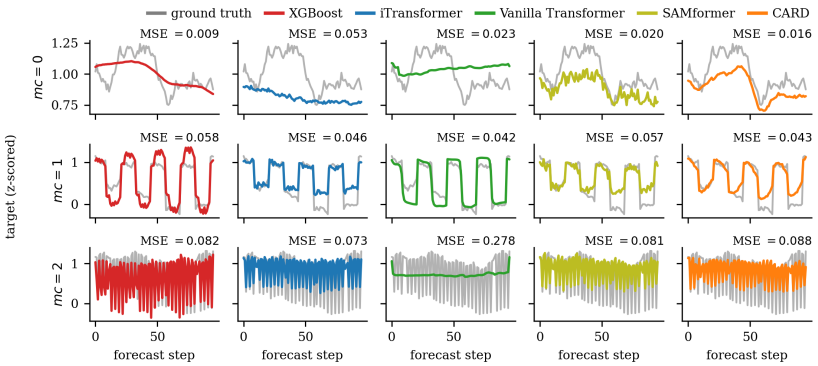


Figure 2: One shared test window by source selection configuration (rows mc=0, 1, 2, $N_J = 1$, $\rho = 0.0$), five models (columns), with the MSE of the plotted window in each panel. Windows are chosen per configuration to be typical for all five models at once (Appendix K). Top row, CARD and SAMformer follow the shape of the delayed source with a level offset, XGBoost gives a smoothed version of it, and iTransformer and VanTF drift toward the window level. Middle row, the plotted forecasts reproduce the daily square-wave pattern. Bottom row, iTransformer, XGBoost, SAMformer and CARD reproduce the rotating alternation with varying fidelity while VanTF produces a nearly constant forecast.

Robustness to source paths and source process. The main grid shares one pair of random-walk sources. We therefore regenerate the mc=0, N_J=1 slice with five additional *source draws*, i.e., new pairs of random-walk source paths generated with different random seeds, and retrain iTransformer, VanTF, and the channel-wise models under the same protocol (Appendix E.1). Across the main-grid source paths and the five new draws, iTransformer's MSE is 0.857–1.580 times persistence and its shape R^2 (Section 3.3) never exceeds 0.215 (Table 6). The failure is therefore not specific to the main realization, whose test block has unusually low variance; raw R^2, unlike shape R^2, changes sharply with that variance. We also replace the random walks with stationary AR(1) sources, which substantially reduce the level shortcut (Appendix E.2). On this nine-cell, three-seed control, VanTF reaches mean $R^2 = 0.920$, while iTransformer remains near the window-mean baseline ($R^2 = -0.021$). *Retrieval is therefore conditional on both the architecture and source process, rather than a universal property of transformer forecasters.*

2. What do the models learn instead, and where is the copy lost? We examine one trained iTransformer on the *reference dataset*, the pure-copy configuration with one white-noise distractor (mc=0, $N_J = 1$, $\rho = 0$), which we use for all single-dataset analyses that follow (one seed, MSE 0.0413, $R^2 = 0.101$).

Attention routing. The target token assigns head-averaged attention weights of 0.160 and 0.202 to source $X^{(1)}$ in the two layers, with 0.293 in the strongest head. Its corresponding self-attention weights are 0.177 and 0.119. This alone does not establish copying.

Input–output dependence. For an exact copy, the Jacobian $\partial \hat{Y}_{t+h} / \partial X_s^{(1)}$ concentrates at the required source index; which is the diagonal $s = h$ in window-relative coordinates. The learned Jacobian (Appendix Figure 3b) places only 1% of squared mass there. Diagonal and off-diagonal entries have the same mean magnitude (0.015), and rows have mean correlation 0.90, indicating similar source aggregation across horizons. Under per-channel instance normalization, the source row sums are exactly zero and the target-history row sums exactly one by construction: normalization removes each input window's level and denormalization restores the target's. The forecast's level can therefore only come from the target's own history; what training fails to learn is the diagonal copy of the source.

Controlled intervention. In every layer, we replace the target token's attention row with weight one on the correct source and zero elsewhere, leaving every other token's attention (including self-attention) as the identity, then retrain on the same dataset and seed (Table 2). This *oracle routing* raises R^2 from 0.101 to 0.801 with normalization unchanged, although the route is supplied

Attention	Instance norm	Encoder LN	MSE	R^2
learned	per-channel	on	0.0413	0.101
learned	shared	off	0.0511	-0.113
oracle	none	on	0.0164	0.642
oracle	per-channel	on	0.0091	0.801
oracle	per-channel	off	0.0092	0.798
oracle	shared	on	0.0048	0.894
oracle	shared	off	**0.0006**	**0.987**
shared-statistics reference			0.0005	0.990
exact solution			0	1

Table 2: Retraining interventions on the reference dataset (mc=0, $N_J = 1$, $\rho = 0$), one seed and 10 epochs. Oracle attention assigns the target token entirely to source channel ($X^{(1)}$) in every layer. Shared instance normalization uses the target window's statistics for all channels. Removing encoder layer normalization improves recovery when combined with shared statistics and oracle routing. Appendix F gives all runs and probes.

rather than learned. Per-channel instance normalization removes relative window offsets needed to reproduce the standardized target (Appendix B). Keeping oracle routing and replacing it with *shared-statistics normalization*, which uses the target window's mean and standard deviation for every channel, reaches 0.894. Layer normalization (Ba et al., 2016) then rescales each token by its own content-dependent standard deviation. Removing encoder layer normalization as well reaches $R^2 = 0.987$. With oracle routing, removing it while retaining per-channel instance normalization leaves R^2 at 0.798, and both normalization changes without oracle routing yield -0.113. With routing left to the model, giving it the relevant statistics does not recover the copy, so missing statistics alone do not explain the failure. Near-exact recovery needs all three changes and the correct route: with shared statistics and no encoder layer normalization, clamping only the target row (other attention learned) gives $R^2 = 0.982$ against 0.988 for the full clamp, and clamping it onto the second source or the distractor gives -0.33 and 0.00. Repeating the best configuration with two additional training seeds gives $R^2 = 0.988$ and 0.987, so this is not a single lucky run (Appendix F).

Locating information loss. We record intermediate representations from the oracle-routed, shared-statistics model and fit a linear readout at each stage (a linear regression from the target token's representation to the true future target, i.e., a linear probe), using blocked cross-validation with a 96-window exclusion interval (Appendix F). Immediately after the first attention block, the target token supports copy recovery at $R^2 = 0.975$; at the encoder output this falls to 0.894. Decomposing layer normalization isolates the decrease: centering and the learned affine transformation preserve 0.975, whereas division alone reduces it to 0.889. Across test windows, the divisor correlates at $+0.97$ with the copied source term's norm and at -0.36 with the target embedding's norm. It therefore rescales the representation according to the amplitude that copying must preserve. Retraining without encoder layer normalization gives $R^2 = 0.991$ for the output probe and 0.984 for the trained prediction head. These measurements *connect the normalization operation to reduced linear recoverability in the examined checkpoint.* This motivates the open question **Q2** (see Section 1).

Channel-wise models. Simple TF + SAM reaches $R^2 = 0.090$ on the reference dataset. Its attention varies little with the query and assigns 0.623 of the target row's weight to the distractor, versus 0.08 to the two sources. The three RevIN-equipped variants reach $R^2 = 0.506$–0.693 on this dataset, although Simple TF + RevIN's worst windows have errors 7–14 times the mean (Appendix G). Within this family, *RevIN thus separates failure from partial recovery, but does not make recovery uniform across windows.*

Source-switching datasets. Predictions largely follow the switching schedule (Figure 2, lower rows). For daily switching (mc=1), a two-level square wave fitted to each forecast window (one level per true gate state) explains $R^2 = 0.907$–0.974 of the examined models' forecasts, compared with 0.889 when fitted to the true target. After subtracting this pattern, only iTransformer and CARD retain residual target correlations above 0.5 at mc=1. VanTF does not reproduce the rotating pattern at mc=2 (Appendix G). We keep this analysis brief because neither switching configuration isolates calendar use (Section 3.4).

3. What is instance normalization doing? Three pairs of channel-wise models differ only in RevIN (Kim et al., 2022): Simple TF vs. Simple TF + RevIN and Spectral TF vs. Spectral TF +

RevIN (the two attention-only pairs), and Simple TF + SAM vs. SAMformer (the SAM pair). The overall MSE ratios with versus without RevIN are 0.295 and 0.281 for the two attention-only pairs, and 0.648 for the SAM pair. RevIN improves every one of the 27 datasets in these comparisons. The mean of the per-dataset ratios over the three pairs increases from 0.259 at mc=0 to 0.479 under rotating switching (i.e., mc=2), indicating a smaller relative benefit. Denormalization restores the current target-window mean and scale to predictions. For drifting random walks, this can provide a useful persistence-like level estimate, consistent with iTransformer's results near the simple reference baselines. The benefit decreases as these statistics become less predictive of future levels. Oracle routing with the original instance normalization still reaches $R^2 = 0.801$, so this stage permits substantial copying while influencing the solution learned under ordinary training.

4. Which inputs do the models rely on? Gated Mixture supplies source channels, an explicit gate and calendar features, as well as target history. To examine their roles, we retrain iTransformer and VanTF on datasets with a single distractor ($N_J = 1$), removing inputs in turn (Appendix H; one seed per condition). **Without the gate channel**, iTransformer achieves MSE 0.052 under daily switching (mc=1) and 0.068 under rotating switching (mc=2), compared with GP errors of 0.070 and 0.055 (at mc=0 the gate is constant, and removing it leaves the MSE at 0.044). It thus beats GP under daily switching but not under rotating switching, and in both cases stays close to its full-input errors (0.054 and 0.065); *the explicit gate is therefore unnecessary for this level of performance.* Additionally holding all timestamp features constant, so that calendar features carry no schedule information, contradicts our prediction of a substantial degradation: daily-switching (mc=1) MSE changes by 0.001, and rotating-switching (mc=2) MSE reaches 0.089, above GP (0.055) but still about three times lower than WM (0.263). This is consistent with the models inferring the switching schedule from the pattern already visible in the lookback window (Section 3.4). **Removing source channels** raises VanTF's MSE to 0.083, 0.237 and 0.329 for mc=0, 1, 2; iTransformer stays at 0.045 on the pure-copy task, and its switching errors rise by 31–65% over the gate-deleted condition (a comparison that also changes gate availability). The remaining target history, gate and distractor still support GP-like persistence, though less accurately than GP itself.

Positive controls. Ridge, XGBoost, CARD and the three RevIN-equipped channel-wise variants outperform persistence on every pure-copy dataset. On Random Causal, which weakens the level shortcut, iTransformer reaches $R^2 = 0.907$ without a distractor but only 0.201 with one Gaussian distractor (three seeds, Appendix I): it can learn retrieval, but not robustly to distractors.

5. Does this transfer beyond synthetic sources? We built one further dataset from real measurements: the public ETTh1 dataset with one added column that copies one of its real channels 96 steps later (Appendix J). Under standard training, both iTransformer ($R^2 = 0.543$) and VanTF ($R^2 = 0.311$) reach substantial positive R^2 here, and iTransformer now outperforms VanTF, consistent with retrieval depending on the source process. The Section 5 intervention reaches $R^2 = 0.978$, close to its synthetic-data result. Whether such gains carry over to ordinary forecasting is **Q3**.

6. Limitations. Our conclusions are conditional on the evaluated architectures, shared training protocol, and controlled data-generating processes; other source processes, delays, horizons, or tuning budgets may yield different outcomes. The main grid uses one source realization, and the real-data check covers one dataset. Finally, the intervention is a diagnostic oracle that localizes failure, not a learned or deployable architectural solution. A complete analysis of the limitations of this work is reported in Appendix L.

6 Conclusion and Future Work

Our findings demonstrate how CTBench can expose weaknesses that remain hidden on natural datasets, support mechanistic diagnosis, and evaluate potential architectural fixes alongside real-world forecasting performance. At present, CTBench focuses on the copy-test pattern. A natural direction for future work is to extend it to additional semantically interpretable forecasting capabilities, thereby creating a broader benchmark that evaluates Transformer architectures from multiple perspectives. Such a benchmark could summarize performance across these capabilities in a strength index, providing the community with a more informative and interpretable tool for comparing and improving Transformer-based forecasters.

REPRODUCIBILITY STATEMENT

The generators are specified in Section 3.1 and Appendix C, the protocol in Section 4, and every table is generated from the verified per-dataset result files using reproducible scripts. A supplementary archive will be published upon acceptance, containing the generator code, the training scripts for every model, the raw per-seed prediction arrays behind every number, our modified iTransformer implementation used for the interventions, with its checkpoints, and the scripts that recompute the Jacobians, attention statistics, ridge and PatchTST columns, seed spreads and baselines. For transparency, Section 5 reports a refuted prediction about calendar features, and Section 3.4 states the limitation of the switching configurations.

AI USE STATEMENT

Large language model tools were used to edit prose, to check tables and figures from the saved result arrays, and to cross-check numbers against those arrays. All experiments, claims and interpretations are the authors' own and were verified by the authors.

ETHICS STATEMENT

Most experiments use only synthetic data. Appendix J additionally uses ETTh1, a public electrical-transformer measurement dataset with no human subjects, personal data, or identifiable information.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

Yu Chen, Nathalia Céspedes, and Payam Barnaghi. A closer look at transformers for time series forecasting: Understanding why they work and where they struggle. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7763–7780, 2025. URL `https://proceedings.mlr.press/v267/chen25f.html`.

Wei Fan, Pengyang Wang, Dongkun Wang, Dongjie Wang, Yuanchun Zhou, and Yanjie Fu. DishTS: A general paradigm for alleviating distribution shift in time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7522–7529, 2023.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

Cyril Garcia and Guillaume Remy. Statistical benchmarking of transformer models in low signal-to-noise time-series forecasting. *arXiv preprint arXiv:2602.09869*, 2026.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Antoine Gonon, Alexandre Cordonnier, and Nicolas Boumal. Gaussian match-and-copy: A minimalist benchmark for studying transformer induction. *arXiv preprint arXiv:2602.07562*, 2026.

Romain Ilbert, Ambroise Odonnat, Vasilii Feofanov, Aladin Virmaux, Giuseppe Paolo, Themis Palpanas, and Ievgen Redko. SAMformer: Unlocking the potential of transformers in time series forecasting with sharpness-aware minimization and channel-wise attention. In *Forty-first International Conference on Machine Learning*, 2024.

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.

Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. In *Advances in Neural Information Processing Systems*, volume 35, pp. 9881–9893, 2022.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. iTransformer: Inverted transformers are effective for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024.

Zhiding Liu, Mingyue Cheng, Zhi Li, Zhenya Huang, Qi Liu, Yanhu Xie, and Enhong Chen. Adaptive normalization for non-stationary time series forecasting: A temporal slice perspective. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

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. *Proceedings of the VLDB Endowment*, 17 (9):2363–2377, 2024.

Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9573–9585, 2020.

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

Xue Wang, Tian Zhou, Qingsong Wen, Jinyang Gao, Bolin Ding, and Rong Jin. CARD: Channel aligned robust blend transformer for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024a.

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

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Yong Liu, Yunzhong Qiu, Haoran Zhang, Jianmin Wang, and Mingsheng Long. TimeXer: Empowering transformers for time series forecasting with exogenous variables. In *Advances in Neural Information Processing Systems*, volume 37, 2024c.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pp. 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, 2022.

Yufa Zhou, Yixiao Wang, Surbhi Goel, and Anru R Zhang. Why do transformers fail to forecast time series in-context? *arXiv preprint arXiv:2510.09776*, 2025.

A Extended Related Work

	Known optimum	Simple ref. baselines	Published forecasters	Intervention inside model	Obstruction identified
Ilbert et al. (2024)	yes	no	one family	optimization only	sharpness (optimization)
Chen et al. (2025)	no	no	several	arch./norm. ablations	normalization, intra-variate structure
Zhou et al. (2025)	yes	no	linear attention	no	theory, no training
Garcia & Remy (2026)	yes	learned only	two-way attention	no	none, transformers win
Gonon et al. (2026)	partial	no	none	no	optimization dynamics
This paper	zero error	PE, WM, GP	twelve	yes	routing, LN division, inst. norm

Table 3: What each diagnostic line supplies. *Known optimum* means the optimal predictor is available in closed form, *simple reference baselines* means every error is reported next to PE, WM and GP (*learned only* means classical learned baselines without these simple reference baselines), *intervention* means the mechanism is tested by changing the model and retraining rather than by inspection alone. Ilbert et al. (2024) retrains a modified attention parameterization to test an optimization account (sharpness). It does not test routing or normalization component-by-component as we do, hence *optimization only*. Chen et al. (2025) also retrains, ablating skip connections, decoder design and normalization on a fixed model family, hence *arch./norm. ablations* rather than *no* intervention. It does not isolate cross-channel routing or measure it with a controlled oracle-attention intervention as we do.

Transformer-based forecasting. Informer (Zhou et al., 2021) selects a sparse set of dominant queries. Autoformer (Wu et al., 2021) combines autocorrelation with trend–seasonal decomposition, while FEDformer (Zhou et al., 2022) performs the decomposition in the frequency domain to reduce computational cost. PatchTST (Nie et al., 2023) represents local temporal patterns as patches and predicts channels independently with shared weights, reducing overfitting in its reported experiments. iTransformer (Liu et al., 2024) instead represents each complete channel as a token, avoiding the combination of heterogeneous measurements within a time-step token. Crossformer (Zhang & Yan, 2023) uses two-stage temporal and dimensional attention; CARD (Wang et al., 2024a) alternates attention over patches and channels. Motivated by a linear task that a transformer can represent but does not learn, SAMformer (Ilbert et al., 2024) combines a shallow channel-wise encoder, sharpness-aware minimization (Foret et al., 2021) and reversible instance normalization (Kim et al., 2022).

What standard benchmarks can and cannot show. Zeng et al. (2023) demonstrate the competitiveness of linear models on standard forecasting benchmarks. Qiu et al. (2024) document evaluation-pipeline inconsistencies and develop a standardized benchmark. These studies highlight different limitations of aggregate comparisons. Without a known optimal predictor, a small performance difference does not identify the forecasting mechanism responsible. Our benchmark adds a controlled setting: the pure-copy task has an exact linear solution, yet none of the evaluated transformer forecasters recovers it to the stated success threshold under our protocol.

Diagnostic and synthetic evaluation. Chen et al. (2025) use model-agnostic mutual-information measures on standard and synthetic datasets to examine intra-variate dependencies, normalization and inter-variate attention. Our zero-Bayes-error tasks support complementary internal probes and retraining interventions. Ilbert et al. (2024) provide a close precedent through a synthetic linear task used to motivate an architectural correction. We instead compare several published forecasters across 27 configurations of source selection and distractor properties. Zhou et al. (2025) prove that linear self-attention cannot outperform the optimal linear predictor on AR(p) data and that recursive predictions decay toward the mean. Gonon et al. (2026) analyze gradient-descent dynamics in a minimal retrieval problem based on second-order correlations, in the context of language-model induction. Garcia & Remy (2026) compare two-way attention transformers, Lasso, boosting and multilayer perceptrons on synthetic processes with known temporal and cross-sectional dependencies. In their low signal-to-noise setting, transformers outperform the classical methods and attention sparsification is especially helpful at high noise levels. The latter two studies are concurrent with ours.

Normalization as mechanism and confound. Liu et al. (2022) describe loss of useful information through excessive stationarization. Fan et al. (2023) distinguish distribution shifts within the lookback from shifts between lookback and forecast periods, and Liu et al. (2023) introduce slice-

adaptive normalization. These approaches recognize that normalization can remove predictive trend and level information. We examine both external instance normalization and internal encoder layer normalization across controlled source selection configurations.

Shortcut learning and mechanistic accounts of copying. Simplicity bias and shortcut learning describe how training can favor easily learned predictive features over the intended dependency (Shah et al., 2020; Geirhos et al., 2020). Transformers can implement copying through induction heads in language modeling (Olsson et al., 2022). Our forecasting experiments examine conditions in which accurate copying is not learned and use retraining interventions to test the roles of routing and normalization.

B Normalization Stages

Reversible instance normalization (Kim et al., 2022) is applied before and after the forecaster. For each channel c in lookback window $\mathcal{W}_t$, it computes the mean $\hat{\mu}^{(c)}$ and standard deviation $\hat{\sigma}^{(c)}$, standardizes the input and reverses this transformation on the output:

$$\tilde{Z}_{s,c} = \frac{Z_{s,c} - \hat{\mu}^{(c)}}{\hat{\sigma}^{(c)}}, \qquad \hat{Y}^{(c)}_{t+h} = \hat{\sigma}^{(c)}\, \tilde{Y}^{(c)}_{t+h} + \hat{\mu}^{(c)}. \tag{4}$$

Statistics are computed separately for each window and channel, with a constant $\varepsilon = 10^{-5}$ added under the square root before division for numerical stability in both implementations we use; this matters in principle at $mc{=}0$, where the gate channel is constant within every window. The gate is normalized by the same per-channel operation as every other input channel, not exempted from it. Because it is exactly constant within the window at $mc{=}0$, that operation maps it to the constant 0 at every step, which removes its information without literally skipping the transform. At $mc{=}1$ and $mc{=}2$ the gate is not constant within every window, so this degeneracy does not apply. The two implementations differ in one respect Equation 4 does not show: iTransformer's own instance normalization applies only the mean/standard-deviation transform above, with no learned affine step, while the channel-wise family's RevIN module additionally applies a learned per-channel affine transform after Equation 4's standardization, with its default initialization (weight one, bias zero). Output denormalization therefore restores each channel's own observed level, while input normalization removes relative channel offsets and scales. iTransformer uses both the mean/standard-deviation operations; the channel-wise family includes matched variants with and without the full RevIN module (transform and its affine step together). *Shared-statistics normalization*, used in Section 5, instead applies the target window's statistics to all channels, preserving relative offsets and scales. The standardized copy relation then includes a window-dependent offset, approximately $0.077/\hat{\sigma}^{(Y)}$. The encoder does not receive $\hat{\sigma}^{(Y)}$ explicitly. Approximating this term by a global constant gives reported MSE 0.00048, corresponding to the $R^2 = 0.990$ reference in Table 2. This reference is not a strict information bound: the intervention includes a unit-variance distractor, whose amplitude after shared normalization can carry information about $1/\hat{\sigma}^{(Y)}$. The near-exact result should therefore be read as a controlled demonstration of capacity under a changed interface, not as a clean estimate of performance without all side channels.

Layer normalization (Ba et al., 2016) acts across the d feature dimensions of each token. Centering subtracts $\mu(x)$; scaling divides by the content-dependent standard deviation $\sigma(x)$. Centering is linear but is not generally information-preserving. In our probe, centering leaves copy recovery unchanged, while division reduces it. With channel-wise tokens, attention can place source information in the target token, whose normalization divisor then depends partly on the copied amplitude. Section 5 measures this dependence and its effect on linear recovery.

C Generator Implementation Details

This section reports full details and parameters about dataset generator implementation[2].

Exact specification. Sources are cumulative sums of independent $\mathcal{N}(0,1)$ increments, with the fixed seed specified in the generator code. The 27 datasets share one source-path pair. Distractors

[2]The Python code of the dataset generators and the datasets used in the evaluation will be released upon acceptance.

are AR(1) processes with stationary variance one and the stated ρ. Observations are hourly from a fixed start date; daily switching selects the second source from noon to midnight. The first 96 target steps are zero-padded. Models use their standard hourly timestamp encoding and training-only standardization. After initialization, $\max_t |Y_t - X^{(1)}_{t-96}| = 0$ at $mc{=}0$ in raw units.

The standardized task. The 96-step delay gives source and target different training means (39.80 and 37.67), with nearly equal standard deviations (27.69 and 27.68). After training-only standardization, the pure-copy relation is $y_t \approx 1.0002\, x^{(1)}_{t-96} + 0.0769$. Using the unrounded coefficients gives a residual below 10^{-14}. The forecasting task in standardized coordinates is therefore an affine copy, not a strict identity.

Variance by split at $mc{=}0$. Over the training windows the z-scored target has variance 0.938, of which 0.917 (98%) is variance of the per-window mean and 0.021 is variance around it. The validation windows have variance 0.016 and the test windows 0.046, of which 66% is between-window. Every headline number of the copy task is therefore measured on a block with one twentieth of the training variance, and checkpoints are selected on a block with one sixtieth of it.

Reference baselines. For the shared source paths and fixed delay, baseline errors depend on the source selection parameter, not on distractor settings. Over 905 test windows, PE and WM are respectively $0.040/0.047$ at $mc{=}0$, $0.576/0.337$ at $mc{=}1$ and $0.452/0.263$ at $mc{=}2$. GP repeats the most recent target value with the forecast step's gate state and gives 0.040, 0.070 and 0.055. A gate-aware window mean averages observed values with that state and gives 0.047, 0.093 and 0.074. Both use the deterministic future gate schedule. Pure-copy persistence error is below its expected value under unit-variance increments (0.064 averaged over horizons), illustrating dependence on the source realization.

Daily-switching control. Fixing all timestamp features changes iTransformer's daily-switching MSE from 0.052 to 0.053 (Table 10), contrary to the recorded prediction of a substantial deterioration. The plotted models, including XGBoost, reproduce the daily square wave (Figure 7). This configuration remains useful as a switching control, although it cannot isolate calendar use. Differences from rotating switching can then be examined without attributing every error to switching itself.

D Evaluated Models

Table 4 records the tokenization, channel-mixing mechanism, normalization choice, and number of runs for every evaluated model. It makes the architectural differences behind the main comparison explicit.

Model	Tokens	Mixing	Inst. norm	Chan. indep.	Runs
Ridge	none	linear map	no	no	1, deterministic
XGBoost	none	trees	no	no	1
VanTF	point	projection	no	no	3 seeds
iTransformer	channel	attention	yes	no	3 seeds
Simple TF and its three variants	channel	attention	switch	no	3 seeds
Spectral TF and Spectral TF + RevIN	channel	attention	switch	no	3 seeds
CARD	patch	dual attn.	yes	no	3 seeds
PatchTST	patch	none	yes	yes	3 seeds

Table 4: The evaluated models, classified by token scope, by how they mix channels, by whether they apply instance normalization (a switch in the channel-wise family, which gives controlled pairs), and by whether they treat each channel independently.

Model	mc=0 (pure-copy)	mc=1 (daily)	mc=2 (rotating)	Grand mean
Ridge (flattened window)	**0.000**	**0.031**	0.080	**0.037**
XGBoost	0.011	0.061	0.090	0.054
Simple TF + RevIN	0.023 ±.002	0.042 ±.001	0.066 ±.002	0.043
SAMformer	0.019 ±.003	0.056 ±.003	0.079 ±.001	0.051
Spectral TF + RevIN	0.013 ±.001	0.056 ±.001	0.087 ±.001	0.052
iTransformer	0.045 ±.004	0.054 ±.003	0.067 ±.002	0.055
CARD	0.020 ±.005	0.052 ±.003	0.103 ±.004	0.059
PatchTST	0.047 ±.003	0.081 ±.002	0.098 ±.004	0.075
Simple TF + SAM	0.042 ±.001	0.076 ±.002	0.120 ±.002	0.079
Vanilla Transformer	0.039 ±.012	0.051 ±.014	0.252 ±.010	0.114
Simple TF	0.162 ±.044	0.096 ±.009	0.184 ±.029	0.148
Spectral TF	0.095 ±.011	0.206 ±.015	0.253 ±.019	0.185
Gate-aware persistence (GP)	0.040	0.070	**0.055**	0.055
Window mean (WM)	0.047	0.337	0.263	0.216
Persistence (PE)	0.040	0.576	0.452	0.356

Table 5: Mean test MSE on the z-scored target per source-selection configuration (9 datasets each) and over the full 27 datasets, lower is better. **Bold** marks the lowest entry in each column, simple reference baselines included. Every attention model is the mean over three seeds, and the small ± value is the across-seed standard deviation of one dataset, averaged over that configuration's nine datasets. Ridge and XGBoost are single deterministic or canonical runs. PatchTST is scored on 896 of the 905 test windows (its loader drops the last partial batch, Appendix E.5). PE repeats the last observed target value, WM repeats the lookback mean, and GP repeats the last lookback target value observed under the same gate state as the forecast step, using the true gate schedule (Section 3.3). At mc=0 GP equals PE. The means repeat the *mean* rows of Table 1, here with the across-seed spread added.

E Alternative results by source selection Configuration and Seed Variation

E.1 Independent source draws

Every main-grid result (Table 1) is conditional on one pair of random-walk source paths. To measure this dependence, we regenerate the mc=0, N_J=1 slice with five additional generator seeds, redraw the sources and distractor, and keep the lag, split, and training protocol fixed. We retrain iTransformer, VanTF, and all six channel-wise variants with one model seed per new draw (a *source draw* is one independently generated realization of the two sources and the distractor). Table 6 displays the five additional draws for iTransformer, VanTF, and three RevIN-equipped channel-wise variants; the other three channel-wise variants were run but are omitted from this compact table. Its entries are R^2 and shape R^2, not MSE relative to persistence. Including the original main-grid realization, these experiments cover six source paths in total. The shared draw, i.e. the main-grid realization, has unusually low test-target variance (0.046, the mc=0 value given in Section 3.3); the variance of the five additional draws range from 0.110 to 0.964 (caption of Table 6). Across all six draws, iTransformer stays within 0.857–1.580 times persistence and its shape R^2 never exceeds 0.215. The three displayed RevIN-equipped channel-wise variants beat persistence with positive shape R^2 on every additional draw. VanTF is notably unstable across draws. These results support the qualitative iTransformer finding while making clear that cell-level rankings remain source-path dependent.

E.2 Stationary-source control

To remove the drifting-level shortcut, we repeat all nine mc=0 cells with independent, unit-variance AR(1) sources ($\rho = 0.9$) in place of random walks. All other generator settings and the shared training protocol remain fixed. Table 7 aggregates three model seeds. VanTF reaches mean $R^2 = 0.920$, while iTransformer remains near the window-mean baseline. This control demonstrates that the random-walk result should not be read as a universal statement about transformer architectures or as evidence that level tracking alone explains every failure.

	iTransformer		VanTF		Simple+RevIN		SAMformer		Spectral+RevIN	
Source Draw	R^2	R^2_{shape}	R^2	R^2_{shape}	R^2	R^2_{shape}	R^2	R^2_{shape}	R^2	R^2_{shape}
101	−0.100	0.021	−12.488	−1.309	0.593	0.541	0.525	0.461	0.726	0.639
102	0.669	−0.048	0.938	0.605	0.890	0.662	0.903	0.693	0.910	0.700
103	0.325	−0.009	−2.392	−0.001	0.709	0.605	0.637	0.466	0.799	0.604
104	0.254	−0.072	−2.493	−0.729	0.628	0.508	0.585	0.424	0.743	0.624
105	0.548	0.153	−0.143	0.176	0.726	0.474	0.696	0.389	0.823	0.679

Table 6: Independent source draws, mc=0 and $N_J = 1$. The table displays the five additional source paths for iTransformer, VanTF, and three RevIN-equipped channel-wise variants; all six channel-wise variants were retrained, but the remaining three are omitted here. Entries average three distractor-autocorrelation cells at model seed 42 and report R^2 and shape R^2. Test target variances for draws 101–105 are approximately 0.182, 0.964, 0.397, 0.110, and 0.171.

Model	MSE	R^2	MSE / WM
VanTF	**0.078**	**0.920**	**0.07**
SAMformer	0.429	0.562	0.41
Spectral TF + RevIN	0.447	0.544	0.42
Simple TF + RevIN	0.477	0.513	0.45
Simple TF + SAM	0.728	0.256	0.69
Spectral TF	0.802	0.181	0.76
Simple TF	0.815	0.168	0.77
iTransformer	0.999	−0.021	0.95
Ridge	$< 10^{-11}$	1.000	0.00
Window mean (WM)	1.055	−0.078	1.00
Persistence (PE)	1.898	−0.939	1.80

Table 7: The copy task on stationary AR(1) sources, all nine mc=0 cells ($N_J \in \{1, 3, 5\}$, $\rho \in \{0, 0.5, 0.9\}$), three seeds, the shared budget of Section 4. Every entry is the mean over the nine cells. Mean-reverting sources leave the training-window level share at 17% against 98% on the random walks of Table 1, so the level shortcut is much weaker by construction, though not eliminated. Persistence is a poor reference on a mean-reverting process, so the binding simple reference baseline here is the window mean (WM) and skill is reported against it. Ridge is exact in every cell. The point-wise VanTF substantially recovers the copy ($R^2 = 0.920$, below the near-exact threshold of Section 3.3) and the channel-wise iTransformer is last, at the window-mean (WM) baseline (Appendix E.2).

E.3 Learning rate and epoch sweep

The main grid trains every model with one fixed learning rate and epoch count. To check whether iTransformer's reference-dataset result is simply a matter of undertraining or a poorly chosen learning rate, we retrained it on the same reference dataset (mc=0, N_J=1, ρ=0) at four learning rates, for 20 and 100 epochs, under both the implementation's default decaying schedule and a schedule that holds the learning rate constant. We repeated the same sweep for the point-wise Vanilla Transformer.

We found that the default schedule halves the learning rate every epoch, which caused iTransformer's best-validation checkpoint to lock onto epoch 1 in every run at this learning rate and dataset: training for 100 epochs under that schedule tests the same epoch-1 checkpoint as training for 20, not a more-trained model. The constant-learning-rate runs below avoid this problem.

The learning rate and training schedule therefore do not explain iTransformer's failure on this dataset: across every tested combination it stays far below near-exact recovery, while the same training choices let a different architecture solve the task. This is consistent with the reference-dataset intervention (Section 5), where the fix that works is a change to routing and normalization, not to training length.

Learning rate (constant, 20 epochs)	iTransformer R^2	Vanilla Transformer R^2
10^{-4}	0.070	**0.967**
5×10^{-4}	0.167	0.939
10^{-3}	-0.198	0.581

Table 8: Test R^2 on the reference dataset (mc=0, $N_J = 1$, $\rho = 0$) under a constant learning rate, one seed. iTransformer's best result across this entire sweep, including the default decaying schedule at up to 100 epochs and four learning rates, is $R^2 = 0.171$; it never approaches the near-exact criterion under any tested setting. Vanilla Transformer solves the task once trained with a constant, sufficiently small learning rate.

E.4 Ridge regression

Ridge predicts the 96 future target values from the flattened 96-step multivariate window, including target history but excluding calendar features. It uses the common split, training-only standardization and 905 test windows. Validation selects the penalty from $\{10^{-3}, \ldots, 10^{3}\}$: 10^{-3} at mc=0, 100 at mc=1, and 1 or 100 at mc=2. A fixed penalty of one gives configuration means 0.0001, 0.030 and 0.082, and overall mean 0.0374, close to 0.0373 with selection. Thus the main result is insensitive to this choice. On every pure-copy dataset, R^2 rounds to 1.000 at three decimals, showing that the shared preprocessing permits essentially exact prediction.

E.5 PatchTST

PatchTST uses the common protocol with its released 3-layer patch configuration. Its loader omits the last partial batch, leaving 896 rather than 905 test windows. On that subset, PE is 0.040, 0.555, 0.440, and WM is 0.047, 0.330, 0.257 for mc=$0, 1, 2$, respectively. These baseline errors differ by at most 4% from the full-window values; fine comparisons should account for the mismatch. PatchTST's pure-copy MSE is 0.047 ($R^2 = -0.024$), near window mean, consistent with its lack of source access. Its daily-switching error (0.081) exceeds those of iTransformer, CARD and the RevIN-equipped family; its rotating-switching error (0.098) exceeds iTransformer's and GP's but is below CARD's (0.103).

F Retraining Interventions and Layer-Wise Probes

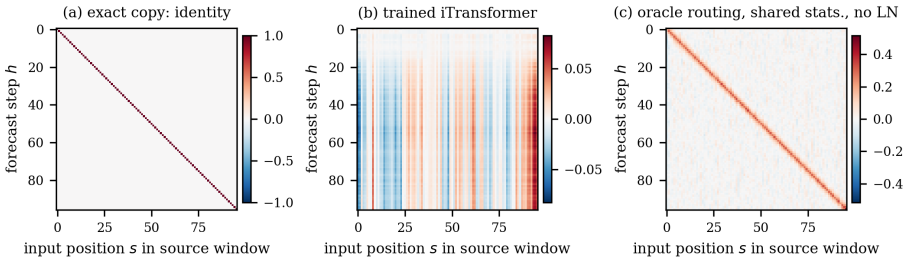


Figure 3: Input-output Jacobian of the forecast with respect to the source window on the reference dataset, averaged over nine test windows. (a) The exact copy is the identity. (b) The trained iTransformer has no diagonal, its columns are nearly constant across the horizon, and its its source row sums are zero by construction. (c) With oracle routing, shared-statistics instance normalization and no encoder layer normalization, the same architecture recovers a banded diagonal with total sensitivity 0.89.

Intervention runs use our modified iTransformer implementation on the reference dataset (mc=0, $N_J = 1$, $\rho = 0$), one seed and 905 test windows. They retain the architecture in Section 4 and use 10 epochs with the implementation's step learning-rate schedule unless specified otherwise. Every

row in Table 2 is a 10-epoch run at this seed. Oracle attention replaces each layer/head attention matrix with the identity except for the target row, which assigns weight one to source $X^{(1)}$.

Does the final result repeat across seeds? The best row of Table 2 (oracle routing, shared statistics, no encoder layer normalization) uses one training seed. We repeated that exact configuration with two more seeds and obtained $R^2 = 0.988$ and $R^2 = 0.987$. All three runs land within 0.001 of each other, so the recovery is a repeatable property of this configuration, not a single favorable initialization.

Layer-wise probes. We record each intermediate token and fit a ridge readout from the target token to the normalized future target window. Probes use either training-window fitting with test-window evaluation (transfer), or blocked five-fold cross-validation within the test block (in-block). The latter excludes 96 windows on each side of each held-out fold to limit overlap. Table 9 reports in-block results for oracle routing and shared statistics, with and without encoder layer normalization. With layer normalization, the first divisor correlates at $+0.97$ with the attention output norm, -0.36 with the target embedding norm, and $+0.82$ with source amplitude relative to target amplitude. It varies by a factor of 2.6 across windows. Relative variation is 0.05 for the embedding term and 0.35 for the attention term, indicating that the copied source strongly influences the divisor.

Stage (target token)	with encoder LN	without encoder LN
embedding, before any attention	0.016	
after first attention block	0.975	0.975
centering only	0.975	
centering and learned affine, no division	0.975	
per-token division only	0.889	
after first layer normalization	0.889	
after first layer	0.894	0.986
after second attention block		0.991
encoder output (what the head sees)	0.894	0.991
the model's own trained head	0.879	0.984
source token's embedding (control)	0.978	

Table 9: Blocked-cross-validation R^2 of linear copy probes at successive target-token representations, using oracle routing, shared statistics and 10 training epochs. Division in layer normalization reduces linear recoverability; without layer normalization, the subsequent blocks improve it. Probe scores report the best of nine ridge penalties on the same folds, so they are descriptive rather than held-out estimates. The trained-head row is not cross-validated: it scores the model's own linear projector on the full test block in the standardized target space, whereas Table 2 scores its de-normalized predictions (hence 0.879 here versus 0.894 there).

G Channel-Wise Models on the Reference Dataset

For Simple TF + SAM on the reference dataset (mc=0, $N_J = 1$, $\rho = 0$), target-row attention is averaged over the saved training snapshots, batches, heads and three seeds. The distractor receives mean weight 0.623 (individual seeds: 0.666, 0.569, 0.634), compared with 0.08 for the two sources. The RevIN variants reach $R^2 = 0.506$ (Simple TF + RevIN), 0.637 (SAMformer) and 0.693 (Spectral TF + RevIN). Higher average accuracy does not ensure uniform performance. Simple TF + RevIN exceeds five times its mean window error on 9–12 windows per seed; these form two or three contiguous events because adjacent windows overlap. Its worst error is 7–14 times the mean. Simple TF has seven such windows in three events on one seed and none on the other two, but its mean error is four to six times larger. Tail behavior must therefore be interpreted alongside absolute error.

Separating switching patterns from source detail. For each model and switching configuration, we use the reference distractor settings and fit a two-level square wave to every forecast window, with one level per true gate state. We report the fit's mean R^2 and correlate the forecast residual with the target residual after the same procedure. This measures source detail beyond the switching pattern. Under daily switching, the fit explains 0.957 of iTransformer's forecast variance, 0.974 for VanTF, 0.929 for CARD, 0.948 for XGBoost, and 0.907–0.956 for the channel-wise family, versus 0.889 for the target. Residual correlations for iTransformer, CARD, VanTF and XGBoost are respectively

0.560, 0.646, 0.230, 0.218 at mc=1, and 0.339, 0.337, 0.082, 0.139 at mc=2. Neural values are three-seed means; XGBoost uses one run. VanTF's rotating-pattern fit has $R^2 = 0.004$, and its mean MSE over the nine rotating-switching datasets, 0.252 (Table 1), is near the window-mean baseline, 0.263.

H Retraining Ablations

Table 10 reports each ablation's test MSE as a mean over the three ρ datasets at $N_J = 1$, and Table 11 gives the same 54 runs by dataset. Freezing timestamps on top of gate deletion moves iTransformer from 0.068 to 0.089 at the rotating gate and leaves VanTF unchanged there (0.247 to 0.241), and deleting the sources raises iTransformer's error by 65% at mc=1 and 31% at mc=2 relative to the gate-deleted run.

	iTransformer			Vanilla Transformer		
	no gate	no gate/calendar	no sources	no gate	no gate/calendar	no sources
mc=0	0.044	0.043	0.045	0.029	0.015	0.083
mc=1	0.052	0.053	0.086	0.046	0.076	0.237
mc=2	0.068	0.089	0.089	0.247	0.241	0.329

Table 10: Retraining ablations on the datasets with $N_J = 1$. Each entry is the mean test MSE over the three ρ datasets from single-seed runs, against PE/WM baselines of $0.040/0.047$, $0.576/0.337$ and $0.452/0.263$ for mc=$0, 1, 2$. *No gate* deletes the explicit gate column, *no gate/calendar* additionally fixes all timestamp features, and *no sources* deletes both source channels. The 54 runs cover these three conditions only, so entries are compared with each other and with the baselines.

config		reference		iTransformer			Vanilla Transformer		
mc	ρ	PE	WM	−gate	−g−d	−src	−gate	−g−d	−src
0	0.0	0.040	0.047	0.043	0.044	0.046	0.027	0.013	0.092
0	0.5	0.040	0.047	0.045	0.041	0.045	0.038	0.017	0.085
0	0.9	0.040	0.047	0.045	0.045	0.045	0.023	0.015	0.072
1	0.0	0.576	0.337	0.054	0.051	0.084	0.049	0.067	0.248
1	0.5	0.576	0.337	0.050	0.054	0.086	0.046	0.092	0.164
1	0.9	0.576	0.337	0.052	0.054	0.088	0.043	0.070	0.299
2	0.0	0.452	0.263	0.071	0.087	0.087	0.240	0.240	0.322
2	0.5	0.452	0.263	0.064	0.088	0.088	0.243	0.243	0.358
2	0.9	0.452	0.263	0.067	0.094	0.093	0.257	0.240	0.307

Table 11: Per-dataset test MSE behind Table 10, datasets with $N_J = 1$, 54 single-seed retraining runs. *−gate* deletes the gate column, *−g−d* additionally freezes every timestamp, *−src* deletes both source channels.

I Random Causal Generator: Implementation and Results

The generator. At each time step, an event occurs independently with probability p. Formally, the source is

$$X_t \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(p), \qquad p = 0.1,$$

where $X_t = 1$ represents a unit-amplitude event lasting one time step and $X_t = 0$ denotes inactivity. The target reproduces each event after a fixed delay $\ell = 96$:

$$Y_t = \begin{cases} 0, & t < \ell, \\ X_{t-\ell}, & t \geq \ell. \end{cases}$$

Thus, an event in the source at time t produces an identical target event at time $t + \ell$, with no observation noise. A second configuration adds an independent Gaussian distractor,

$$J_t \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_J^2), \qquad \sigma_J = 0.9,$$

which is approximately three times the source's empirical standard deviation but has no relationship to the target. Neither configuration includes calendar features. The lookback length, forecast horizon, delay, data split, and number of evaluation windows match those of Gated Mixture.

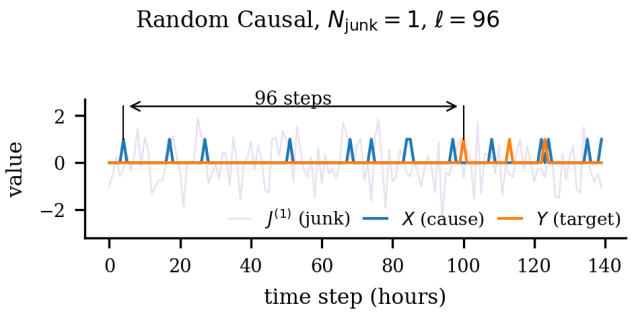


Figure 4: The Random Causal task. The cause channel (blue) is a sparse random pulse train and the target (orange) repeats it 96 steps later. One distractor channel (pale lavender) is shown for scale.

Results. Without random-walk level variation, this generator provides a complementary test of retrieval. Table 12 reports a three-seed mean under the same shared protocol as the main grid (Section 4): 20 epochs, the shared architecture for iTransformer and VanTF, and PatchTST's own released configuration. PatchTST performs near window mean in both conditions. iTransformer reaches $R^2 = 0.907$ without a distractor. Adding one Gaussian distractor with three times the source standard deviation lowers iTransformer to $R^2 = 0.201$. VanTF reaches $R^2 = 0.426$ clean and 0.416 with the distractor; under three seeds these two numbers are close enough that the distractor has no clear effect on VanTF (an earlier single-seed run had suggested the opposite direction). The large iTransformer deterioration with a distractor present, and its absence for VanTF, both replicate under the fair protocol.

	clean		one Gaussian distractor	
Model	MSE	R^2	MSE	R^2
iTransformer	**0.091**	**0.907**	0.788	0.201
Vanilla Transformer	0.566	0.426	**0.576**	**0.416**
PatchTST	1.030	-0.045	0.997	-0.011

Table 12: The Random Causal task, test MSE on the z-scored target over 905 windows (test variance 0.986). Three-seed mean under the shared 20-epoch protocol. The distractor condition adds one Gaussian channel with three times the pulse channel's standard deviation.

Specification and training budget. The source is active 9.8% of the time. Its raw standard deviation is 0.30, versus 0.91 for the distractor. Target standardization uses training mean 0.099 and standard deviation 0.298; evaluated test-window variance is 0.986. The analytic copy rule is exact in raw units everywhere, including at initialization. The implemented reference's MSE 0.0003 instead comes from standardizing the source and target channels with their own training-set statistics (mean 0.102, standard deviation 0.303 for the source versus 0.099 and 0.298 for the target), the same source-versus-target offset as Section 3.2; a same-scale copy that does not correct for this small affine mismatch lands 0.0003 off the standardized target, and Table 12 reports that measured value rather than rounding it to zero. Table 12 uses three seeds and the same 20-epoch protocol as the main grid. iTransformer and VanTF use the shared architecture; PatchTST uses its own released configuration (the same one used for its column in the main grid).

J What the Diagnostics Suggest for Real Data

A copy task built from real measurements. Every result so far uses synthetic sources. To check whether the same intervention also helps outside synthetic data, we built one further dataset from ETTh1, a public hourly record of electrical-transformer measurements. We kept all seven of its real

channels, including oil temperature (OT), unchanged and added one new column: the value of one real load channel, High UseFul Load (HUFL), delayed by 96 steps. This keeps the exact-answer property of the rest of this paper (the correct forecast is still fully contained in the input window) while giving the source channel real measurement statistics instead of an artificial random walk. We trained iTransformer and VanTF on this dataset under the shared protocol (three seeds), and separately reran the intervention from Section 5 on it (one seed, 10 epochs, matching Appendix F's protocol).

Model/condition	Epochs	Seeds	MSE	R^2	R^2_{shape}
iTransformer	20	3	0.203	$0.543_{\pm 0.005}$	0.701
Vanilla Transformer	20	3	0.306	$0.311_{\pm 0.074}$	0.406
Shared statistics, LN off	10	1	0.188	0.578	0.857
Oracle, shared statistics, LN off	10	1	0.010	0.978	0.983

Table 13: Delayed HUFL copy added to ETTh1. Standard runs use three seeds and 20 epochs. Intervention runs use seed 42 and ten epochs. R^2 shows seed SD where replicated. Different budgets prevent a clean effect estimate between standard and intervention rows. All use 905 test windows. PE and WM have $R^2 = -0.932$ and -0.158, respectively.

Three things stand out. First, both architectures reach substantial positive R^2 under standard training here, unlike on the synthetic random-walk grid where iTransformer stays near the window-mean baseline; because PE and WM are themselves both worse than a constant prediction on this dataset (negative R^2), a positive R^2 here is not explained by these two baselines, and shape R^2, which gives no credit for level tracking at all, is higher than raw R^2 for every row, which establishes within-window prediction; without a source ablation on this dataset, however, it does not establish that the standard models use the copied channel. Second, iTransformer actually does better than VanTF on this dataset, the opposite ordering from the random-walk pure-copy configuration. This is not a contradiction of our earlier results: this paper's own explanation for the synthetic-grid gap is that recovery depends on the source process, not on a fixed ranking of architectures, and a real measurement channel is a different source process from a random walk or a stationary AR(1) series. Third, applying the intervention to this dataset reaches $R^2 = 0.978$ (shape $R^2 = 0.983$), close to the 0.987 obtained on synthetic data. The normalization change alone, without forcing routing, also raises both R^2 and shape R^2 over the standard-training mean; we do not treat the size of that gap as a precise effect estimate, since the two runs differ in seed count and training length. On the synthetic reference dataset, the same normalization change without oracle routing instead lowers R^2 (from 0.101 to -0.113, Table 2), so the effect of this component depends on the setting. So the mechanism identified in Section 5 is not confined to synthetic sources, though this is one real-data anchor at one delay and one channel, not a general claim about forecasting on real datasets.

Further hypotheses, still untested. Beyond this one anchor, the synthetic findings suggest further hypotheses for real data. Restoring window statistics may explain why RevIN improves standard benchmark results (Kim et al., 2022) and why its benefit decreases when distributions change (Chen et al., 2025): the operation helps when current statistics predict future levels. Level tracking is consistent with strong linear and persistence-based baselines (Zeng et al., 2023) and with theoretical decay toward the mean (Zhou et al., 2025). Limited source routing is also consistent with the small contribution of inter-variate attention on self-dependent datasets (Chen et al., 2025). Our layer-normalization probes identify a possible mechanism to investigate in non-stationary settings. These broader explanations still require further real-data interventions beyond the one anchor above.

K Additional Figures

For each source selection configuration, Figure 2 uses a shared window chosen to be representative of the five plotted models. Indices 866, 203 and 717 minimize the sum of squared relative differences between a window's error and each model's mean window error. Figure 5 provides separate model panels for the pure-copy window. Figures 6–8 show each model's lowest- and highest-error windows among 905 test windows. The worst pure-copy transformer forecasts mainly have level offsets, while XGBoost remains closer to the target. Under daily switching, errors primarily concern the square

wave's amplitude and level. Under rotating switching, VanTF is nearly constant even on its best window; the other plotted models retain switching but often underestimate amplitude.

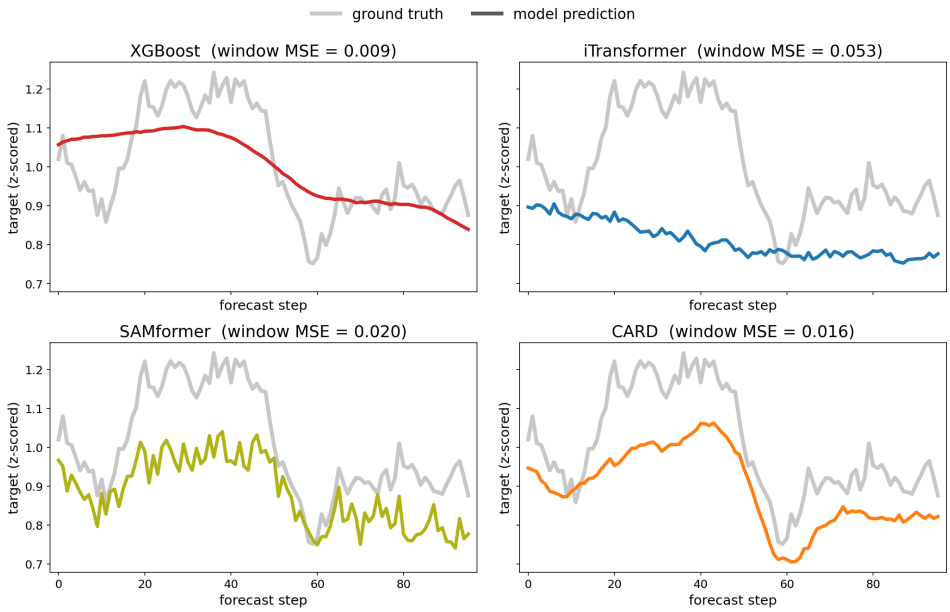


Figure 5: Per-model panels of the mc=0 window of Figure 2, with the MSE of the plotted window. CARD and SAMformer follow the shape of the delayed source with a downward level offset, XGBoost gives a smoothed version of it, and iTransformer stays below the target and does not follow its shape.

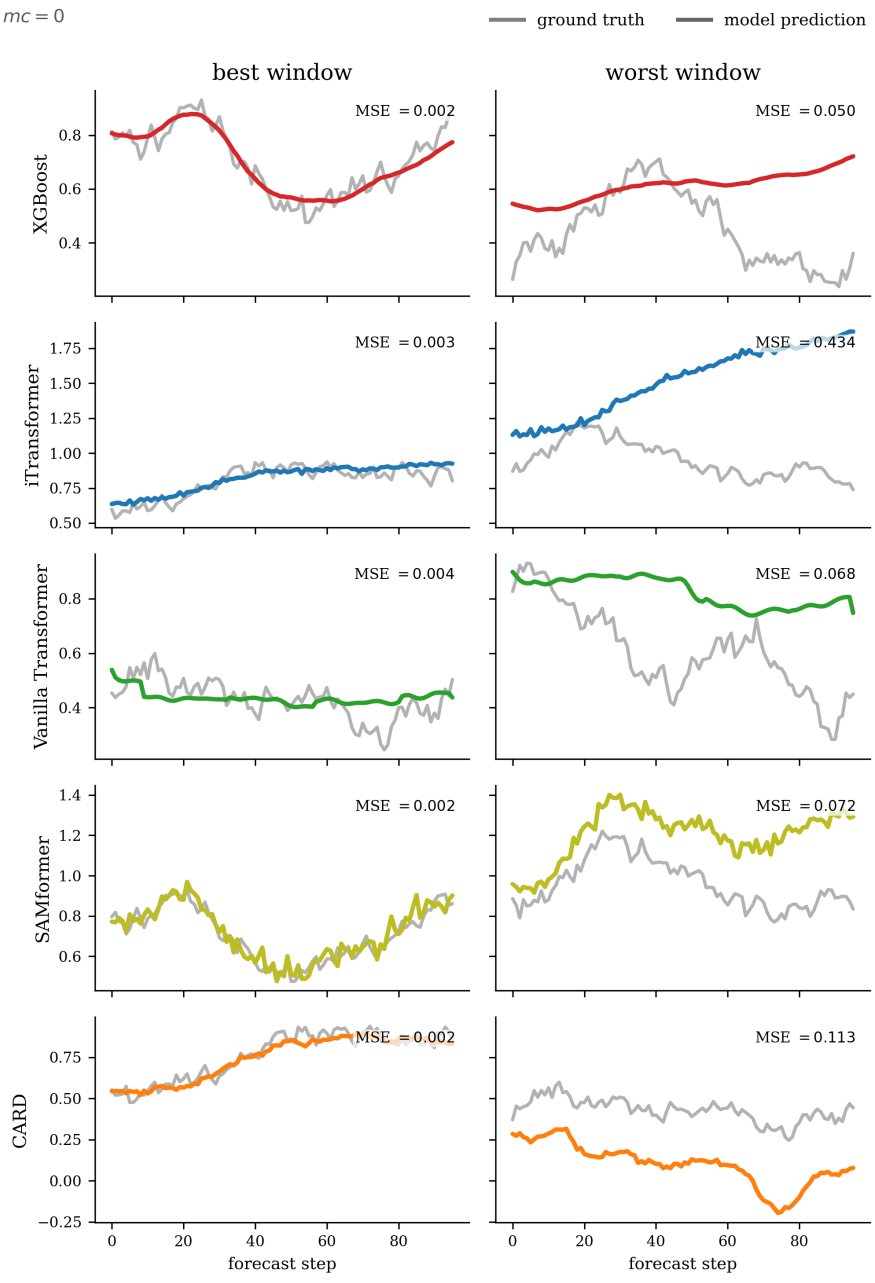


Figure 6: Best and worst test window per model at mc=0, selected by per-window MSE over the 905 test windows. The worst transformer windows are dominated by level offsets.

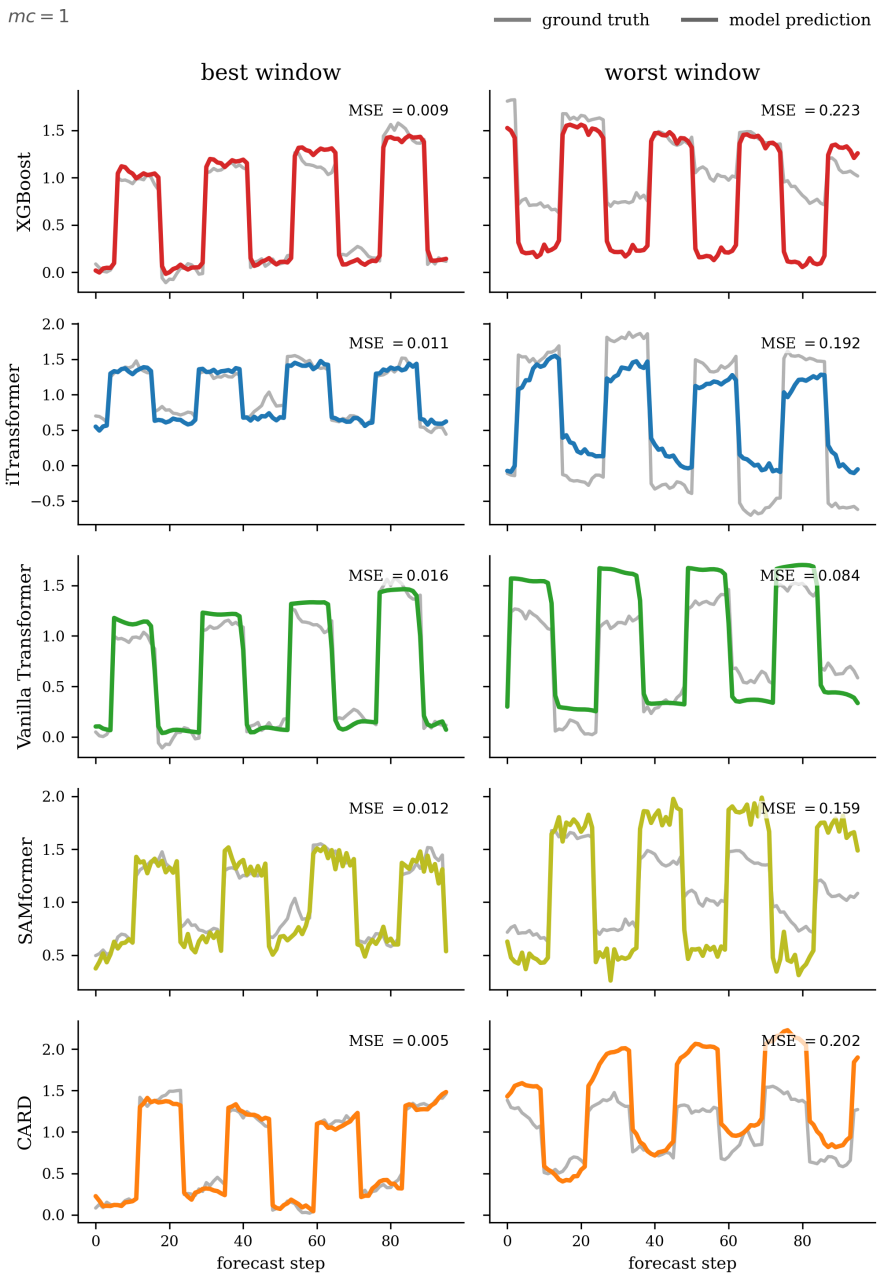


Figure 7: Best and worst test window per model at the daily gate. The plotted models reproduce the daily square wave; their errors mainly concern amplitude and level.

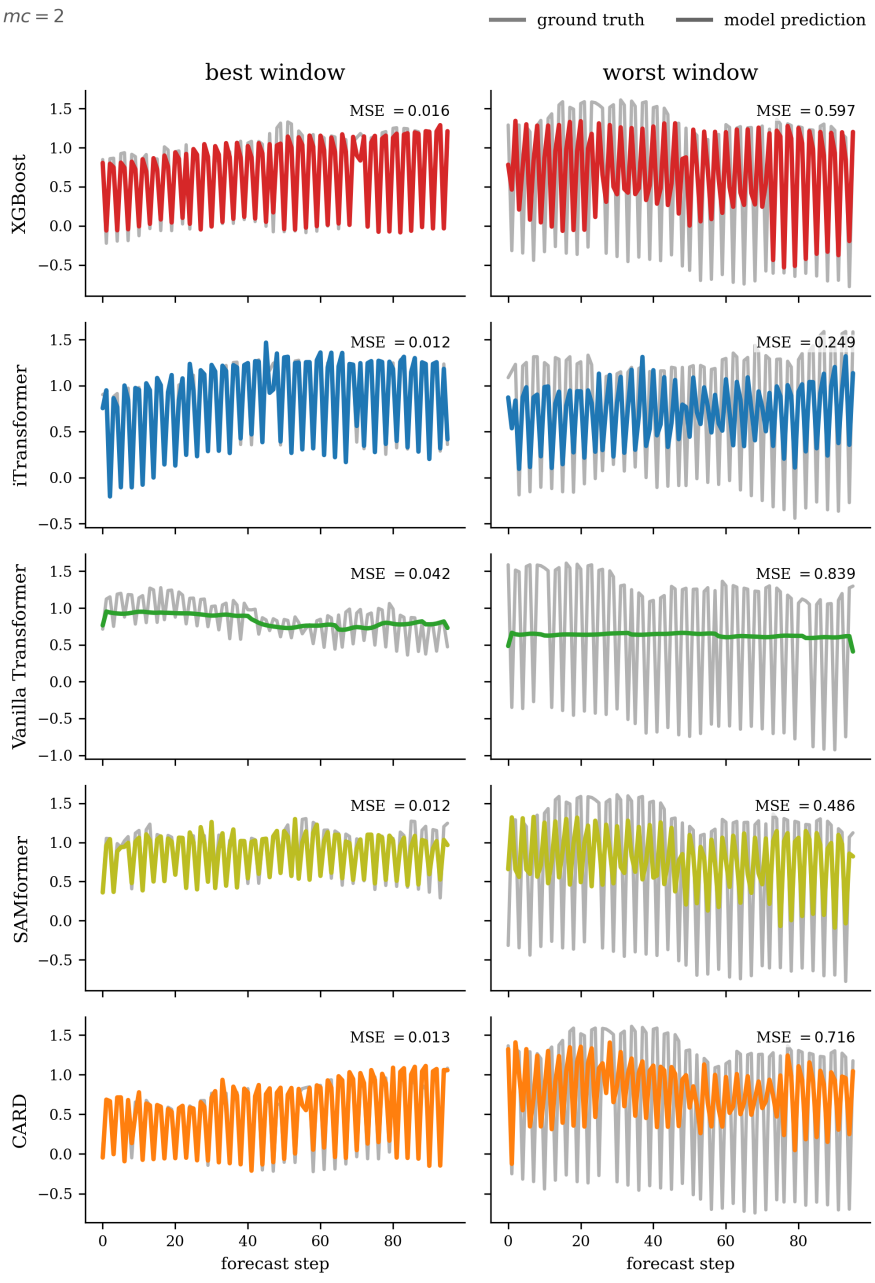


Figure 8: Best and worst test window per model at the rotating gate. VanTF is flat in both columns, while the remaining models keep the alternation and lose amplitude in their worst windows.

L LIMITATIONS IN FULL

(1) *Shared source realization in the full grid.* All 27 Gated Mixture datasets use one pair of source random walks. Appendix E.1 adds five independent draws on the mc=0, N_J=1 slice and shows that the iTransformer conclusion and the partial recovery of RevIN-equipped channel-wise models persist there. It does not rerun CARD, XGBoost, or PatchTST, and it cannot establish that every cell-level ranking generalizes to new paths.

(2) *Source processes favor level tracking.* Random-walk levels explain 98% of training-window variance, and the validation block has one sixtieth of that variance. Both favor a simple level predictor. The stationary AR(1) control in Appendix E.2 substantially reduces this shortcut and changes the model ordering, but it covers only pure-copy and one AR coefficient. Random Causal provides a second control, built from a different generator, now trained under the same shared protocol as the main grid. These experiments do not exhaust the source processes or validation splits on which retrieval may be learned.

(3) *One temporal setting.* Experiments use $L = H = \ell = 96$ and $T = 5{,}000$, with 5, 7, or 9 observed channels in Gated Mixture. Retrieval always uses one fixed delay aligned with the lookback boundary. Other horizons, delays and training sizes remain untested.

(4) *Restricted switching mechanisms.* The gate is binary, deterministic and available both as a channel and through calendar features. Neither switching configuration isolates calendar use (Section 3.4). Real changes can be latent, gradual and noisy. The target has no additional observation noise or separately imposed trend or seasonality.

(5) *Overlapping evaluation windows.* Adjacent test windows share 95 of 96 steps. The number of independent events is therefore much smaller than 905, and several high-error windows can represent one event.

(6) *Limited intervention scope.* Near-exact recovery is demonstrated on one dataset at a ten-epoch budget; the best configuration repeats within 0.001 R^2 across three seeds on that dataset (Appendix F) and, at one seed, on the ETTh1-derived dataset (Appendix J), but every other row of Table 2 is single-seed. Routing is supplied by clamping attention rather than learned, and in every reported oracle-attention run every other token's attention row, including self-attention, is also set to the identity. This isolates the oracle-routed target row from a fully learned baseline, but it does not separate the effect of correcting routing from the effect of removing the rest of learned attention, so the improvement from 0.101 to 0.801 should be read as a property of this combined intervention, not of routing alone. Clamping only the target-token row, with every other row learned, and wrong-source clamps are reported in Section 5; whether channel-identity embeddings let the model learn correct routing on its own is not tested. The closest evidence toward a fully learned correction is the ETTh1 row in Appendix J, where the normalization fix alone, with routing left to the model, raises R^2 from 0.543 to 0.578 without reaching the oracle-routed result (0.978). A correction that also learns routing end to end remains future work. Shared-statistics normalization changes the original input/output interface and may expose target-scale information through the unit-variance distractor, so $R^2 = 0.987$ concerns a modified system with potential side channels. Removing layer normalization may affect training stability in other settings. Attention and Jacobian probes describe the examined checkpoints, not a distribution of independently trained models.

(7) *Model and validation scope.* Conclusions concern the evaluated neural forecasters under a shared, untuned budget. PatchTST uses its released 3-layer configuration and 896 test windows. Random Causal (Appendix I) uses the same shared 20-epoch, three-seed protocol as the main grid. Appendix J adds one real-data anchor (ETTh1), but this covers one dataset, one delay and one copied channel, so broader real-data validation, decoder-style and pretrained model comparisons, per-model hyperparameter searches, an explicitly exogenous-input architecture such as TimeXer (Wang et al., 2024c), and a learned gate-conditioned predictor for the switching configurations all remain outside this study.