TIME DECODE: A DENOISING CODEBOOK-BASED MODEL FOR TIME SERIES ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Time-series models have been widely explored for learning temporal representations from sequential data. However, real-world time series often contain complex noise from measurement errors, stochastic disturbances, and environmental perturbations, which can affect modeling at multiple stages. We identify three interrelated noise dimensions: data-level noise in token representations, objective-level noise in pretraining targets, and inference-level noise in model predictions. To address these issues, we propose **Time DeCode**, a denoising codebook-based time-series model that connects representation learning and forecasting inference through a learned code space. Specifically, a noise-aware discrete tokenizer separates sparse irregular components from observations and encodes temporal structure into discrete codes. A next multi-code prediction objective learns temporal dependencies by predicting the code indices of future patches rather than their continuous values, reducing reliance on noisy numerical targets. For prediction, code-space prediction calibration aggregates overlapping predictions for the same future position before decoding to improve inference stability. Extensive experiments on forecasting and classification benchmarks demonstrate the effectiveness of Time DeCode, while noise injection evaluations further support its performance and robustness under noisy observations. The code is available at `https://anonymous.4open.science/r/Time_Decode/`.

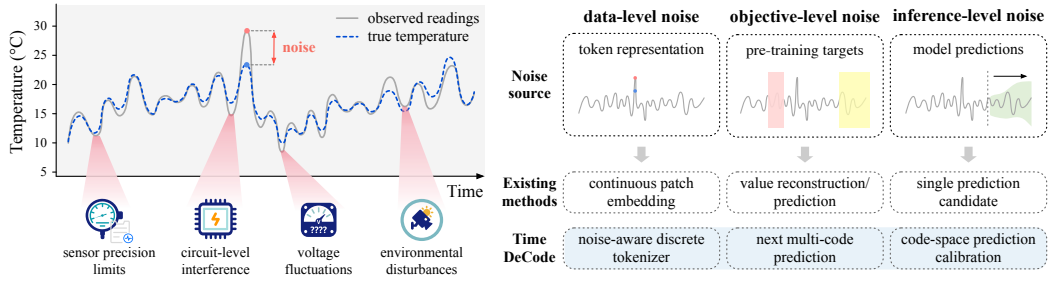
1 INTRODUCTION

Time series analysis plays a central role in modeling dynamic systems across many real-world applications, including financial analysis, traffic flow modeling, and energy demand estimation (Wen et al., 2022; Li et al., 2018; Zhou et al., 2021). Recent Transformer-based time-series models have been widely explored for learning representations from time-series data, where a model is typically pretrained with a self-supervised objective to better capture complex temporal patterns from unlabeled sequences (Dong et al., 2024; Liu et al., 2024b; Das et al., 2024; Cheng et al., 2026; Nie et al., 2022).

However, these works often assume that observed time series are clean and reliably reflect the temporal signals. In practice, time series are often intertwined with complex noise, such as measurement errors, stochastic disturbances, and environmental perturbations (Kostelich & Schreiber, 1993; Hassani et al., 2025; Ma et al., 2023). As illustrated in Figure 1(a), in a sensor-based temperature monitoring system, the observed readings deviate from the underlying temperature trajectory at certain time points due to sensor precision limits, circuit-level interference, voltage fluctuations, and environmental disturbances. Directly learning from such noisy readings may make the model capture noise-specific variations rather than the underlying temporal patterns. This highlights the need for *time-series models that learn meaningful temporal representations from noisy real-world data*. However, this goal is nontrivial, as noise can affect time-series modeling through three interrelated dimensions: data-level noise, objective-level noise, and inference-level noise.

Data-level noise arises from how noisy observations are converted into token representations. Existing methods typically divide a continuous time series into patches and directly project each raw patch into a continuous embedding space to obtain a patch token (Nie et al., 2022; Cheng et al., 2026). Because these tokens are derived from raw continuous values, their representations are sensitive to noise-induced variations in the observed values. As a result, patches with similar underlying temporal patterns may be represented differently due to noisy observations. A key requirement is thus

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



(a) Noise in real-world sensor temperature time series. **(b) Three dimensions of noise propagation in time series learning.**

Figure 1: Noise in real-world time series and the corresponding designs of Time DeCode.

to abstract noisy observations into compact and noise-tolerant token representations that preserve stable temporal patterns.

Objective-level noise emerges when noisy observations serve as the targets of pretraining objectives. Existing time-series pretraining methods typically use the observed values of masked patches or future segments as targets for continuous-value reconstruction or prediction (Dong et al., 2023; Li et al., 2023b; Liu et al., 2024b; Das et al., 2024). Because these targets are defined in the original numerical space, noise in the observations directly affects the pretraining loss. Consequently, the model may learn to match non-informative numerical variations rather than stable temporal patterns, which reduces pretraining robustness and efficiency. A desired pretraining objective should reduce reliance on noisy continuous targets and provide a more robust learning signal.

Inference-level noise concerns the influence of noisy temporal contexts on model predictions at inference time. At inference time, existing methods typically produce a single prediction candidate from the given historical context (Nie et al., 2022; Liu et al., 2024a; Zeng et al., 2023). Such one-shot prediction can be fragile: when the input sequence contains noise, the resulting prediction may be biased toward incidental variations in the observations, allowing input perturbations to affect the final result. A more robust inference mechanism is therefore required to reduce dependence on any individual prediction made under noisy contexts.

To address noise throughout time-series modeling, we propose **Time DeCode**, a denoising codebook-based time-series model. The key designs of Time DeCode compared with existing methods are summarized in Figure 1(b). Time DeCode first learns a *noise-aware discrete tokenizer* that maps continuous time series segments into a finite set of discrete code embeddings, reducing sensitivity to data-level noise and encouraging the model to capture transferable token representations. Building on the learned discrete representations, Time DeCode further reformulates the pretraining task from continuous value prediction to *next multi-code prediction*: predicting the code indices of multiple future patches rather than their continuous values, which can make the supervisory signal less sensitive to objective-level noise. For forecasting, Time DeCode further introduces *code-space prediction calibration*, which combines overlapping predictions for the same future position through learnable aggregation of their code embeddings before decoding. This design improves inference stability by reducing the influence of inference-level noise.

Our main contributions are summarized as follows:

- We identify noise propagation as a fundamental limitation of existing time series modeling paradigms, and systematically characterize it from three dimensions: data-level noise, objective-level noise, and inference-level noise.
- We propose Time DeCode, a denoising codebook-based time-series model that connects noise-aware discrete tokenization, next multi-code prediction, and code-space prediction calibration through a learned code space.
- Extensive experiments on time series forecasting and classification benchmarks, together with noise injection evaluations, demonstrate the effectiveness and robustness of the proposed method.

2 Related Work

Multivariate Time Series Analysis. Multivariate time series analysis has been studied using diverse architectures for capturing temporal dependencies. Transformer-based forecasting meth-

ods (Zhou et al., 2021) use attention mechanisms to model long-range temporal interactions. Subsequent designs incorporate decomposition (Wu et al., 2021), frequency-domain modeling (Zhou et al., 2022), patch-level representations (Nie et al., 2022), and variate-centric attention (Liu et al., 2024a). Linear- and MLP-based methods (Zeng et al., 2023; Li et al., 2023a; Ekambaram et al., 2023) offer lightweight alternatives through normalization, decomposition, and temporal mixing. Although these architectural advances improve temporal modeling, directly embedding continuous observations can still entangle informative patterns with incidental fluctuations, leaving the resulting representations sensitive to observational noise.

Self-Supervised Time Series Representation Learning. Self-supervised pretraining enables time-series models to learn temporal representations from unlabeled sequences. Common objectives include masked reconstruction (Nie et al., 2022; Dong et al., 2023; Li et al., 2023b; Cheng et al., 2026) and future prediction (Liu et al., 2024b; Das et al., 2024). Large-scale pretrained models (Goswami et al., 2024; Ansari et al., 2024; Das et al., 2024) further explore learning across diverse time-series corpora. These methods use different forms of supervision: reconstruction and regression objectives operate on continuous values, whereas Chronos (Ansari et al., 2024) quantizes numerical values into tokens and learns their distributions using cross-entropy. However, supervision derived from noisy observations may retain incidental fluctuations, either in numerical targets or in their assigned discrete tokens. Consequently, self-supervised objectives can encourage fitting observational variations that contribute little to learning meaningful temporal dependencies.

Discrete Tokenization for Time Series. Vector quantization provides a mechanism for mapping continuous features to discrete codebook entries. VQ-VAE (Van Den Oord et al., 2017) learns discrete latent representations through reconstruction, while VQGAN (Esser et al., 2021) extends this approach to high-quality image synthesis. For time series, TOTEM (Talukder et al., 2024) learns a VQ-VAE-based tokenizer for forecasting, imputation, and anomaly detection across diverse domains. HDT (Feng et al., 2025) combines an l_2 normalization enhanced vector quantization strategy with a hierarchical Transformer to formulate multivariate forecasting as discrete token generation. Despite their representational benefits, discrete codebooks do not inherently distinguish informative temporal structure from observational noise. When tokenization is optimized to reconstruct noisy inputs, irregular fluctuations may be retained in the learned codes, while perturbations near quantization boundaries can alter code assignments. Such effects can carry observational noise into subsequent code-based learning and prediction.

3 METHODOLOGY

In this section, we present **Time DeCode**, a denoising codebook-based time-series model. As illustrated in Figure 2, Time DeCode consists of three components to address the three noise dimensions: (1) a *noise-aware discrete tokenizer* that reduces sensitivity to data-level noise; (2) *next multi-code prediction* that replaces noise-sensitive continuous supervision with structured discrete code prediction; and (3) *code-space prediction calibration* that stabilizes inference by aggregating multiple predictions for the same future position.

Problem Formulation. Let $\mathbf{X} \in \mathbb{R}^{T \times C}$ denote an observed multivariate time series with T time steps and C variables. We model the observations as

$$\mathbf{X} = \mathbf{X}^\star + \boldsymbol{\epsilon}, \tag{1}$$

where $\mathbf{X}^\star$ represents the underlying temporal signal and $\boldsymbol{\epsilon}$ denotes observational noise. Since $\mathbf{X}^\star$ is not directly observable during training, our goal is to learn informative and noise-robust temporal representations from $\mathbf{X}$ for downstream time-series analysis under noisy contexts.

We partition $\mathbf{X}$ into N_p non-overlapping patches of length P, denoted by $\{\mathbf{x}_i\}_{i=1}^{N_p}$, where $\mathbf{x}_i \in \mathbb{R}^{P \times C}$. Each observed patch can similarly be written as $\mathbf{x}_i = \mathbf{x}_i^\star + \boldsymbol{\epsilon}_i$. For existing methods whose learning objectives are defined over continuous targets, a loss such as $l(\hat{\mathbf{x}}_i, \mathbf{x}_i)$ is evaluated against observations containing both temporal structure and noise. Time DeCode instead learns a code space designed to reduce sensitivity to incidental noise and uses this space for predictive learning and forecast aggregation.

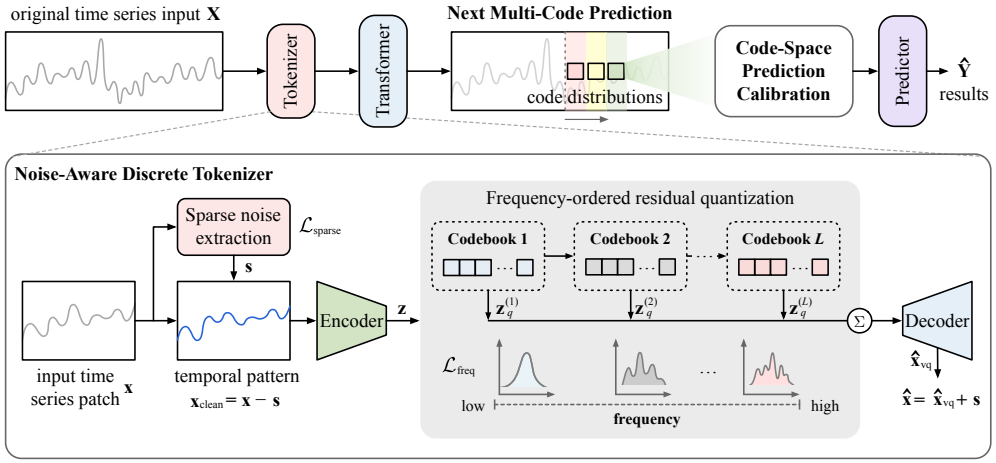


Figure 2: Overview of **Time DeCode**. A noise-aware discrete tokenizer separates sparse irregular components from observed patches and represents the remaining temporal structure using frequency-ordered residual codebooks. A Transformer learns temporal dependencies through next multi-code prediction. For forecasting, code-space prediction calibration aggregates overlapping predicted embeddings for the same future position before decoding.

Noise-Aware Discrete Tokenizer. To address data-level noise, we learn a tokenizer that separates sparse irregular components from observed patches and maps the remaining structure to discrete temporal codes. For an input patch $\mathbf{x}$, a lightweight network S_η estimates an irregular component $\mathbf{s}$:

$$\mathbf{s} = S_\eta(\mathbf{x}), \qquad \mathbf{x}_{\text{clean}} = \mathbf{x} - \mathbf{s}. \tag{2}$$

Here, $\mathbf{x}_{\text{clean}}$ denotes the estimated temporal pattern inferred from the observed patch. By separating sparse irregular variations from the dominant temporal structure, the tokenizer is encouraged to encode representative and transferable patterns into the discrete codebooks.

To prevent the sparse branch from capturing patterns that should be represented by the codebooks, we impose two regularization terms. First, an l_1 penalty $\mathcal{L}_{\text{sparse}} = \|\mathbf{s}\|_1$ discourages it from absorbing the entire input. Second, we encourage the sparse component to be representationally distinct from the patterns stored in the codebooks. Specifically, we map $\mathbf{s}$ into the same embedding space using a frozen shadow encoder $\widetilde{E}_\phi$ and minimize its similarity with the codebook embeddings:

$$\mathcal{L}_{\text{orth}} = \frac{1}{|\mathcal{E}|} \sum_{\mathbf{e} \in \mathcal{E}} \left| \cos\Big(\widetilde{E}_\phi(\mathbf{s}), \text{sg}[\mathbf{e}] \Big) \right|, \tag{3}$$

where $\mathcal{E}$ denotes the embeddings from all codebook layers and $\text{sg}[\cdot]$ denotes stop-gradient. During this regularization, the shadow encoder and codebook embeddings are fixed, so the loss only updates the sparse branch.

To capture temporal patterns at multiple levels of granularity, we employ a stack of L residual codebooks. The encoder first maps the estimated temporal pattern into a latent representation:

$$\mathbf{z} = E_\phi(\mathbf{x}_{\text{clean}}) \in \mathbb{R}^{C \times D}. \tag{4}$$

Each codebook then quantizes the information that remains unexplained by the preceding layers. Specifically, starting from $\mathbf{r}^{(1)} = \mathbf{z}$, the l-th layer selects, for each channel c, the nearest embedding from its codebook $\mathcal{E}^{(l)} = \{\mathbf{e}_k^{(l)}\}_{k=1}^K$:

$$k_c^{(l)} = \arg \min_{k \in \{1,\dots,K\}} \left\| \mathbf{r}_c^{(l)} - \mathbf{e}_k^{(l)} \right\|_2^2, \qquad \mathbf{z}_{q,c}^{(l)} = \mathbf{e}_{k_c^{(l)}}^{(l)}. \tag{5}$$

The selected embedding is then removed from the current residual:

$$\mathbf{r}^{(l+1)} = \mathbf{r}^{(l)} - \text{sg}\left[\mathbf{z}_q^{(l)} \right], \tag{6}$$

allowing subsequent codebooks to represent information not captured by earlier layers. The final discrete representation $\mathbf{z}_q = \sum_{l=1}^L \mathbf{z}_q^{(l)}$ combines the codes from all layers. The decoder then reconstructs the underlying temporal pattern: $\hat{\mathbf{x}}_{\text{vq}} = D_\psi(\mathbf{z}_q)$.

We further encourage the codebook layers to capture complementary temporal structure through a frequency-ordering loss. Each layer's quantized output is decoded separately, and a differentiable frequency score g_l is computed from its normalized Fourier spectrum:

$$\mathcal{L}_{\text{freq}} = \frac{1}{L-1} \sum_{l=1}^{L-1} \text{softplus}(g_l - g_{l+1}), \tag{7}$$

which promotes a progression from coarse temporal structure to finer residual detail across layers. For $L = 1$, we set $\mathcal{L}_{\text{freq}} = 0$. Therefore, the tokenizer is trained with

$$\begin{aligned} \mathcal{L}_{\text{tok}} = \lambda_{\text{rec}} \left\| \mathbf{x} - (\hat{\mathbf{x}}_{\text{vq}} + \mathbf{s}) \right\|_2^2 + \lambda_{\text{vq}} \sum_{l=1}^{L} \beta \left\| \mathbf{r}^{(l)} - \text{sg}[\mathbf{z}_q^{(l)}] \right\|_2^2 \\ + \lambda_{\text{s}} \mathcal{L}_{\text{sparse}} + \lambda_{\text{orth}} \mathcal{L}_{\text{orth}} + \lambda_{\text{freq}} \mathcal{L}_{\text{freq}}. \end{aligned} \tag{8}$$

The original observed patch is reconstructed as $\hat{\mathbf{x}} = \hat{\mathbf{x}}_{\text{vq}} + \mathbf{s}$, so the reconstruction objective does not require ground-truth clean signals. With exponential moving average updates, the vector-quantization term reduces to the commitment loss. After training, each patch $\mathbf{x}_i$ is represented by its quantized embedding $\mathbf{z}_{q,i} \in \mathbb{R}^{C \times D}$ and layer-wise code indices $\mathbf{K}_i = \{k_{i,c}^{(l)}\}_{l,c}$.

Next Multi-Code Prediction. To address objective-level noise, we use tokenizer-assigned code indices as supervisory targets instead of continuous patch values. The tokenizer is frozen during this stage, defining a fixed code space for self-supervised predictive learning. Numerical perturbations that leave the assigned codes unchanged therefore do not alter the supervisory targets, reducing sensitivity to such variations while preserving supervision for temporal learning.

Let $\mathbf{Z}_{1:t} = [\mathbf{z}_{q,1}, \ldots, \mathbf{z}_{q,t}]$ denote the historical code embeddings. A causal Transformer F_θ predicts the codes of multiple future patches at each stage, with separate output heads for the codebook layers. Let M denote the stride between successive prediction stages and N the number of future patches predicted at each stage. At stage s, the model takes $\mathbf{Z}_{1:sM}$ as context and appends N placeholder vectors. The hidden representation $\mathbf{h}_{s,j,c}$ at the placeholder for future patch $sM + j$ and channel c produces a categorical distribution:

$$\boldsymbol{\pi}_{s,j,c}^{(l)} = \text{softmax}(W_l \mathbf{h}_{s,j,c}), \qquad j = 1, \ldots, N, \tag{9}$$

where $\boldsymbol{\pi}_{s,j,c}^{(l)} \in \mathbb{R}^K$ is conditioned on $\mathbf{Z}_{1:sM}$.

The next multi-code prediction objective is

$$\mathcal{L}_{\text{NMCP}} = -\frac{1}{S} \sum_{s=1}^{S} \sum_{l=1}^{L} \alpha_l \sum_{j=1}^{N} \sum_{c=1}^{C} \log \left[\boldsymbol{\pi}_{s,j,c}^{(l)} \right]_{k_{sM+j,c}^{(l)}}, \tag{10}$$

where S is the number of valid stages satisfying $sM + N \leq N_p$, α_l weights the l-th codebook layer, and $[\cdot]_k$ selects the probability assigned to code k. Predicting multiple future patches provides supervision over extended temporal structure. When $M < N$, successive prediction windows overlap, supplying multiple estimates for shared future positions. This overlap supports the prediction aggregation used during inference, as described below.

Code-Space Prediction Calibration. At inference time, predictions conditioned on noisy temporal contexts may reflect incidental variations in the observations. To reduce dependence on individual predictions, Time DeCode aggregates overlapping estimates for the same future position in the learned code space. As illustrated in Figure 3, predicted code distributions are converted into soft embeddings, combined using rank-based learnable gates, and decoded into forecasts.

Hard argmax selection can abruptly change the selected code when competing logits are close. Instead, for each layer l, we convert the predicted logits $\mathbf{o}^{(l)}$ into a temperature-scaled distribution and compute the corresponding expected embedding:

$$\begin{aligned} \boldsymbol{\pi}_\tau^{(l)} &= \text{softmax}\left(\mathbf{o}^{(l)} / \tau \right), \\ \hat{\mathbf{z}}^{(l)} &= \sum_{k=1}^{K} [\boldsymbol{\pi}_\tau^{(l)}]_k \mathbf{e}_k^{(l)}, \qquad \hat{\mathbf{z}} = \sum_{l=1}^{L} \hat{\mathbf{z}}^{(l)}, \end{aligned} \tag{11}$$

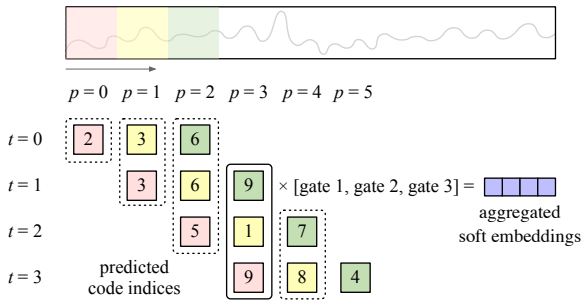


Figure 3: Code-space prediction calibration. Overlapping prediction windows yield multiple estimates for the same future position. Predicted code distributions are converted into soft code embeddings using the learned codebooks, aggregated with normalized rank-based learnable weights, and decoded into forecasts.

where $\tau > 0$ controls the sharpness of the code distribution. Smaller values concentrate probability on the most likely code, while larger values retain information from multiple plausible codebook entries. Position, channel, and prediction-rank indices are omitted for clarity.

For a future patch position p covered by R_p prediction windows, let $\{\hat{\mathbf{z}}_p^{(r)}\}_{r=1}^{R_p}$ denote the predicted embeddings. We aggregate these estimates using rank-based learnable gates:

$$\bar{\mathbf{z}}_p = \frac{\sum_{r=1}^{R_p} w_r \hat{\mathbf{z}}_p^{(r)}}{\sum_{r=1}^{R_p} w_r + \varepsilon_{\text{num}}}, \qquad w_r = \sigma(\gamma_r), \tag{12}$$

where r indexes the prediction rank, γ_r is a learnable scalar, σ is the sigmoid function, and $\varepsilon_{\text{num}} > 0$ ensures numerical stability.

The tokenizer encoder, decoder, and codebooks remain fixed, preserving the learned code space. After aggregation, the future embedding sequence $\bar{\mathbf{Z}}_{\text{pred}}$ is decoded into the forecast:

$$\hat{\mathbf{Y}} = D_\psi\big(\bar{\mathbf{Z}}_{\text{pred}}\big)\,. \tag{13}$$

By combining estimates obtained from different preceding contexts before decoding, calibration reduces reliance on individual predictions and aims to improve prediction stability under noisy inputs.

4 Experiments

In this section, we evaluate Time DeCode on long-term time series forecasting and multivariate time series classification to demonstrate its effectiveness across different time series modeling tasks.

Time Series Forecasting Datasets. We conduct forecasting experiments on seven widely used public benchmarks (Nie et al., 2022): ETTm1, ETTm2, ETTh1, ETTh2, Weather, ECL, and Traffic. These datasets are collected from real-world scenarios and contain measurements of electricity transformer operation, electricity consumption, weather conditions, and road traffic.

Time Series Classification Datasets. We evaluate Time DeCode on ten datasets from the UEA Multivariate Time Series Classification Archive (Bagnall et al., 2018), using the same dataset subset as TimesNet (Wu et al., 2022). These datasets are collected from diverse real-world scenarios, enabling a comprehensive evaluation of the model across different time series classification settings.

4.1 Main Results

In this section, we validate the effectiveness of Time DeCode on both time series forecasting and classification datasets.

Time Series Forecasting. For the forecasting task, we compare Time DeCode with conventional forecasting methods and time series pretraining methods. The former include TimeFilter (Hu et al., 2025), iTransformer (Liu et al., 2024a), PatchTST (Nie et al., 2022), TimesNet (Wu et al., 2022), DLinear (Zeng et al., 2023), FEDformer (Zhou et al., 2022), Informer (Zhou et al., 2021), and Autoformer (Wu et al., 2021). The latter include TOTEM (Talukder et al., 2024), TimeSiam (Dong

Table 1: Average long-term forecasting results with an input length of 96. Each result is averaged over prediction horizons of 96, 192, 336, and 720. The best result is shown in **bold**, and the second-best result is underlined.

Dataset	**Ours**		Chronos-2 (2025)		TimeSiam (2024)		TOTEM (2024)		TimeFilter (2025)		iTransformer (2024)		PatchTST (2023)		TimesNet (2023)		DLinear (2023)		FEDformer (2022)		Informer (2021)		Autoformer (2021)	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	**0.372**	**0.383**	0.392	0.388	0.377	0.396	0.393	0.394	0.383	0.396	0.407	0.413	0.387	0.402	0.406	0.415	0.404	0.408	0.441	0.454	0.832	0.670	0.519	0.498
ETTm2	**0.277**	**0.315**	0.284	0.318	0.280	0.326	0.300	0.337	0.280	0.326	0.291	0.335	0.282	0.329	0.294	0.331	0.355	0.402	0.300	0.345	1.692	0.909	0.320	0.362
ETTh1	**0.409**	**0.419**	0.526	0.481	0.429	0.437	0.474	0.451	0.453	0.437	0.462	0.454	0.452	0.450	0.461	0.455	0.460	0.457	0.462	0.469	1.020	0.787	0.507	0.501
ETTh2	**0.369**	**0.390**	0.387	0.402	0.376	0.403	0.436	0.436	0.391	0.412	0.385	0.408	0.377	0.404	0.410	0.425	0.564	0.519	0.428	0.445	4.594	1.759	0.455	0.469
Traffic	**0.420**	**0.271**	0.493	0.304	0.454	0.289	0.546	0.310	0.424	0.275	0.422	0.282	0.553	0.361	0.629	0.332	0.643	0.400	0.624	0.389	0.777	0.437	0.638	0.395
ECL	0.167	**0.259**	0.201	0.283	0.186	0.271	0.199	0.283	**0.166**	0.263	0.180	0.270	0.215	0.303	0.206	0.304	0.225	0.319	0.235	0.347	0.397	0.465	0.255	0.358
Weather	0.239	**0.264**	0.260	0.268	0.255	0.278	**0.239**	0.270	0.245	0.275	0.264	0.284	0.259	0.281	0.259	0.285	0.265	0.317	0.358	0.406	0.707	0.580	0.323	0.370

et al., 2024), and Chronos-2 (Ansari et al., 2025). To ensure a fair comparison and isolate the effect of each pretraining strategy, all training stages use only in-domain data from the training split of the corresponding target dataset.

With the input length fixed to 96, we evaluate Time DeCode across four prediction horizons, $\{96, 192, 336, 720\}$, and report the results averaged over these horizons in Table 1. Detailed results for individual prediction horizons are provided in Appendix C.1. Time DeCode performs consistently well across the seven benchmarks, achieving the lowest average MAE on all seven datasets and the lowest average MSE on five. Its advantage holds against both categories of baselines: relative to TimeFilter, the strongest conventional forecasting method, it reduces the overall average MSE and MAE by 3.8% and 3.5%, respectively; compared with TimeSiam, the strongest pretraining-based method, the corresponding reductions are 4.4% and 4.1%. These improvements are consistent with the motivation behind our noise-aware representation design, which separates irregular fluctuations from predictive temporal patterns and encodes the latter into discrete codes. This design helps the model focus on stable, informative structures in the historical context, supporting more accurate forecasts across diverse datasets.

Time Series Classification. For the classification task, we compare Time DeCode with Chronos-2 (Ansari et al., 2025), TOTEM (Talukder et al., 2024), iTransformer (Liu et al., 2024a), PatchTST (Nie et al., 2022), Informer (Zhou et al., 2021), TimesNet (Wu et al., 2022), and DLinear (Zeng et al., 2023). We adapt Time DeCode by replacing next multi-code prediction with a masked code prediction objective, which recovers masked code indices from the observed context, and adapting the Transformer backbone to use bidirectional attention for sequence encoding. For downstream classification, the encoded representations are aggregated and passed to a classification head. As shown in Figure 4, Time DeCode achieves the highest mean accuracy of 74.18% across the ten datasets, outperforming TimesNet, the best-performing baseline in this comparison, by 0.44 percentage points. Detailed per-dataset results are provided in Appendix C.2. We attribute this improvement to its noise-aware representations, which suppress irrelevant fluctuations while preserving discriminative temporal patterns. These results demonstrate the applicability of Time DeCode beyond forecasting.

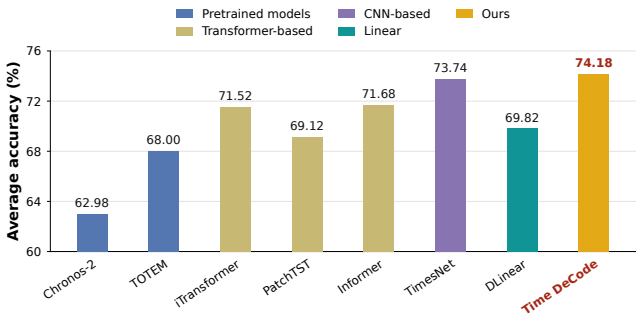


Figure 4: Average classification accuracy across ten UEA datasets.

4.2 Denoising Analysis

Experimental Setting. To examine the effectiveness of Time DeCode under noisy observations, we introduce controlled corruptions at the training and testing stages and evaluate their effects on forecasting performance (see Appendix D for details). At the testing stage, corruptions are applied only to test inputs, while all models use fixed checkpoints without corruption-specific retraining. These corruptions cover five categories: additive perturbations; sparse perturbations represented by outliers; randomly missing observations; block-wise missing observations; and distributional

Table 2: Forecasting MSE under test-input and training-stage corruption on ETTh1, with both input and prediction lengths fixed to 96. Bold and underlined values indicate the best and second-best results within each row, respectively. Results are averaged over severity levels and perturbation seeds within each category. Lower is better; "—" denotes settings not evaluated.

	Time DeCode	TimeFilter	PatchTST	TOTEM	TimeSiam	Chronos-2
(a) Testing stage corruption						
Additive noise	**0.3655**	0.3951	0.3830	0.3862	<u>0.3800</u>	0.4076
Outliers	**0.3638**	0.3948	0.3825	0.3856	<u>0.3798</u>	0.4070
Random missingness	**0.3659**	0.3958	0.3833	0.3865	<u>0.3813</u>	0.4088
Block missingness	**0.3913**	0.4108	0.4051	<u>0.4034</u>	0.4051	0.4372
Level shifts	**0.3692**	0.3970	0.3876	0.3887	<u>0.3836</u>	0.4155
(b) Training stage corruption						
Gaussian noise	**0.3635**	—	0.3812	0.3856	<u>0.3779</u>	0.3893
Outliers	**0.3630**	—	0.3810	0.3823	<u>0.3782</u>	0.4050
Random missingness	**0.3643**	—	0.3812	0.3853	<u>0.3779</u>	0.3976

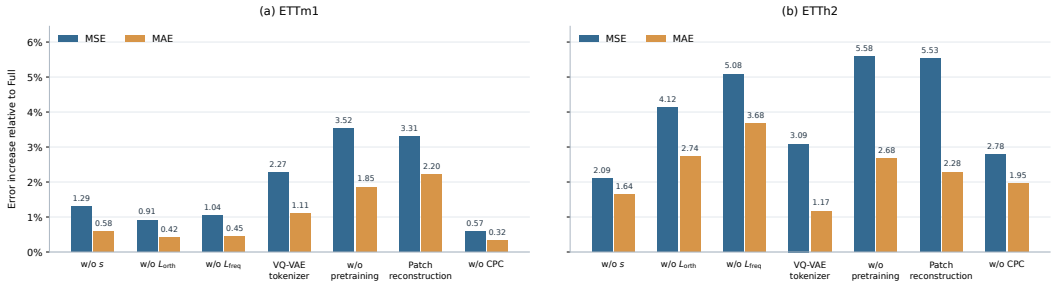


Figure 5: Ablation study on (a) ETTm1 and (b) ETTh2. Bars show the percentage increase in MSE and MAE relative to the corresponding Full model. Positive values indicate worse performance.

changes represented by level shifts. At the training stage, we separately introduce Gaussian noise, outliers, and random missingness into the pretraining context, then evaluate the resulting models on uncorrupted test data. The resulting models are subsequently evaluated on original test data to determine whether they can learn predictive temporal representations from corrupted training contexts. For all experiments, both the input length and prediction horizon are fixed to 96.

Results and Analysis. As shown in Table 2, Time DeCode achieves the lowest MSE across all evaluated testing and training stage corruption categories, indicating that the learned representations retain useful forecasting information under diverse forms of input noise. Across the five testing stage categories, Time DeCode obtains an average MSE of 0.3711, reducing the error of the strongest baseline, TimeSiam, by 3.84%. Under training stage corruption, Time DeCode achieves an average MSE of 0.3636, compared with 0.3780 for the strongest baseline, corresponding to a relative reduction of 3.81%. These results demonstrate the robustness of Time DeCode to corruptions at both the training and testing stages, supporting its ability to learn informative temporal representations from noisy data and maintain forecasting accuracy under noisy observations.

4.3 Model Analysis

Ablation Study. We examine the contributions of three key elements of Time DeCode: noise-aware discrete tokenizer, next multi-code prediction pretraining, and code-space prediction calibration (CPC). For the tokenizer, we remove the noise component $\mathbf{s}$ (w/o $\mathbf{s}$), the orthogonality loss $\mathcal{L}_{\text{orth}}$ (w/o $\mathcal{L}_{\text{orth}}$), and frequency-ordering loss $\mathcal{L}_{\text{freq}}$ (w/o $\mathcal{L}_{\text{freq}}$) individually, and further replace the proposed tokenizer with a standard codebook method (VQ-VAE tokenizer). To study the pretraining strategy, we consider a variant without pretraining (w/o pretraining) and another that replaces token prediction with patch-value reconstruction (patch reconstruction). Finally, we remove CPC to evaluate the contribution of code-space prediction calibration (w/o CPC). These seven variants are evaluated on ETTm1 and ETTh2, with both the input length and prediction horizon fixed to 96. Figure 5 reports the percentage increases in MSE and MAE relative to the full model.

The full model consistently outperforms all seven variants on both datasets in terms of MSE and MAE, confirming that each proposed component contributes to the final performance. On ETTh2,

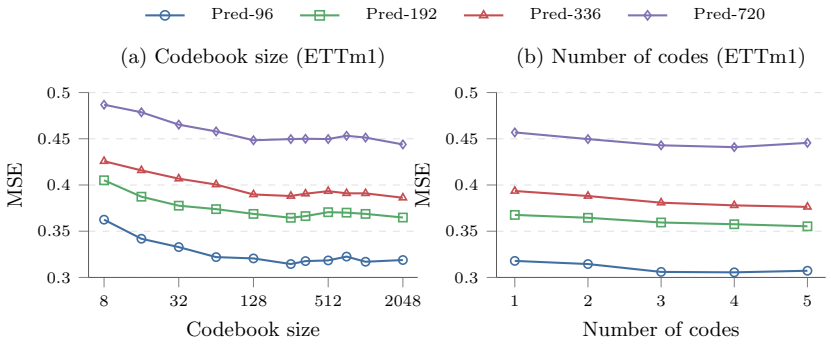


Figure 6: Parameter sensitivity analysis. (a) Codebook size and (b) number of codes on ETTm1 under four prediction horizons. The legend applies to panels (a) and (b). Vertical scales differ across panels.

removing pretraining and replacing discrete token prediction with patch-value reconstruction increase MSE by 5.58% and 5.53%, respectively, demonstrating that the performance gain arises not only from pretraining itself but also from aligning its objective with discrete temporal representations. The tokenizer ablations provide further support for our noise-aware representation design: replacing the proposed tokenizer with a standard VQ-VAE or removing $\mathbf{s}$, $\mathcal{L}_{\text{orth}}$, or $\mathcal{L}_{\text{freq}}$ consistently degrades forecasting performance. In addition, the performance drop observed after removing CPC confirms the benefit of aggregating overlapping predictions in the code space. Overall, these results validate the complementary contributions of three key components to the performance of Time DeCode.

Parameter Sensitivity. To further examine how the degree of discretization affects forecasting performance, we vary the codebook size and the number of residual codebook layers in residual quantization. Experiments are conducted on ETTm1 under prediction horizons of $\{96, 192, 336, 720\}$.

As shown in Figure 6(a), increasing the codebook size substantially reduces MSE when the codebook is small, suggesting that sufficient capacity is needed to represent diverse temporal patterns. Beyond approximately 128-256 entries, improvements become smaller and vary across prediction horizons. For example, a codebook size of 256 yields the lowest MSE for the 96-step horizon, whereas 2048 performs best for the 336- and 720-step horizons. However, the improvements are not monotonic, indicating that a larger codebook does not necessarily yield better forecasts. These results suggest that moderate codebook sizes capture much of the predictive structure, while additional capacity offers horizon-dependent benefits.

Figure 6(b) examines the use of one to five residual codes. Increasing the number of codes from one to four reduces MSE across all four prediction horizons. This is consistent with residual quantization progressively refining the discrete representation and retaining information that a single code may miss. Adding a fifth code yields smaller, mixed effects: MSE decreases slightly for the 192- and 336-step horizons but increases for the 96- and 720-step horizons. The results indicate diminishing returns from additional codes, with four codes achieving the lowest MSE averaged across the evaluated horizons.

5 Conclusion

In this paper, we examine how noise affects time-series representation learning and subsequent inference through three interrelated dimensions: data-level noise in token representations, objective-level noise in pretraining targets, and inference-level noise in model predictions. We propose Time DeCode, a denoising codebook-based time-series model that addresses these issues. Time DeCode combines noise-aware discrete tokenization to capture informative temporal structure, next multi-code prediction to learn temporal dependencies from discrete supervision, and code-space prediction calibration to aggregate overlapping predictions before decoding during inference. Extensive experiments on time series forecasting and classification benchmarks, together with noise injection evaluations, demonstrate the effectiveness and robustness of the proposed method.

GenAI Usage Disclosure

During the preparation of this work, the authors used generative AI tools for language polishing and writing refinement. All technical content, experimental design, analysis, and conclusions were reviewed and verified by the authors.

References

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, et al. Chronos-2: From univariate to universal forecasting. *arXiv preprint arXiv:2510.15821*, 2025.

Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018, 2018. URL `https://arxiv.org/abs/1811.00075`.

Mingyue Cheng, Xiaoyu Tao, Zhiding Liu, Qi Liu, Hao Zhang, Rujiao Zhang, and Enhong Chen. TimeMAE: Self-supervised representations of time series with decoupled masked autoencoders. In *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining*, pp. 498–508, 2026.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 10148–10167, 2024.

Jiaxiang Dong, Haixu Wu, Haoran Zhang, Li Zhang, Jianmin Wang, and Mingsheng Long. SimMTM: A simple pre-training framework for masked time-series modeling. *Advances in Neural Information Processing Systems*, 36:29996–30025, 2023.

Jiaxiang Dong, Haixu Wu, Yuxuan Wang, Yun-Zhong Qiu, Li Zhang, Jianmin Wang, and Mingsheng Long. Timesiam: A pre-training framework for siamese time-series modeling. In *International Conference on Machine Learning*, pp. 11412–11436. PMLR, 2024.

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

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

Shibo Feng, Peilin Zhao, Liu Liu, Pengcheng Wu, and Zhiqi Shen. Hdt: Hierarchical discrete transformer for multivariate time series forecasting. *arXiv preprint arXiv:2502.08302*, 2025.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. *arXiv preprint arXiv:2402.03885*, 2024.

Hossein Hassani, Leila Marvian Mashhad, Manuela Royer-Carenzi, Mohammad Reza Yeganegi, and Nadejda Komendantova. White noise and its misapplications: Impacts on time series model adequacy and forecasting. *Forecasting*, 7(1):8, 2025.

Yifan Hu, Guibin Zhang, Peiyuan Liu, Disen Lan, Naiqi Li, Dawei Cheng, Tao Dai, Shu-Tao Xia, and Shirui Pan. Timefilter: Patch-specific spatial-temporal graph filtration for time series forecasting. *arXiv preprint arXiv:2501.13041*, 2025.

Eric J Kostelich and Thomas Schreiber. Noise reduction in chaotic time-series data: A survey of common methods. *Physical Review E*, 48(3):1752, 1993.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SJiHXGWAZ.

Zhe Li, Shiyi Qi, Yiduo Li, and Zenglin Xu. Revisiting long-term time series forecasting: An investigation on linear mapping. *arXiv preprint arXiv:2305.10721*, 2023a.

Zhe Li, Zhongwen Rao, Lujia Pan, Pengyun Wang, and Zenglin Xu. Ti-mae: Self-supervised masked time series autoencoders. *arXiv preprint arXiv:2301.08871*, 2023b.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. In *International conference on learning representations*, volume 2024, pp. 11116–11140, 2024a.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. *arXiv preprint arXiv:2402.02368*, 2024b.

Peitian Ma, Zhen Liu, Junhao Zheng, Linghao Wang, and Qianli Ma. Ctw: confident time-warping for time-series label-noise learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 4046–4054, 2023.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

Sabera Talukder, Yisong Yue, and Georgia Gkioxari. Totem: Tokenized time series embeddings for general time series analysis. *arXiv preprint arXiv:2402.16412*, 2024.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

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

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. *arXiv preprint arXiv:2210.02186*, 2022.

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.

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

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

A Datasets and Experimental Setup

A.1 Time Series Forecasting Datasets

Dataset Statistics. We evaluate Time DeCode on seven public forecasting benchmarks used in PatchTST (Nie et al., 2022), covering electricity, weather, and transportation. Table 3 summarizes their main characteristics.

- **ETT.** The Electricity Transformer Temperature datasets contain oil temperature and six power-load variables from two electricity transformers (Zhou et al., 2021). ETTm1/2 are sampled every 15 minutes, while ETTh1/2 are sampled hourly.
- **Weather.** Weather contains 21 meteorological variables, including temperature and humidity, recorded every 10 minutes at a weather station in Germany.
- **ECL.** The Electricity Consumption Load dataset contains hourly electricity consumption measurements from 321 clients.
- **Traffic.** Traffic records hourly road occupancy rates from 862 sensors on freeways in the San Francisco Bay Area.

Data Splits and Preprocessing. We split each dataset chronologically into training, validation, and test sets. For the ETT datasets, we use consecutive periods of 12, 4, and 4 months, respectively, following the benchmark implementation, which treats each month as 30 days. For Weather, ECL, and Traffic, we use a 70%/10%/20% split.

Each variable is standardized using the mean and standard deviation estimated from the training split, and the same transformation is applied to the validation and test data. Forecasting samples are constructed using sliding windows with an input length of 96 and prediction horizons of $\{96, 192, 336, 720\}$. Validation and test inputs may include preceding historical observations, while their prediction targets remain entirely within the corresponding split.

A.2 Time Series Classification Datasets

Dataset Statistics. We evaluate Time DeCode on ten multivariate time series classification datasets from the UEA archive, following the benchmark selection of TimesNet (Wu et al., 2022). These datasets span physiological signals, motion trajectories, speech recordings, spectral measurements, and traffic observations. Table 4 summarizes their main characteristics.

- **Physiological signals.** FaceDetection distinguishes face images from scrambled images using magnetoencephalography (MEG) recordings. SelfRegulationSCP1 and SelfRegulationSCP2 contain electroencephalography (EEG) signals collected during binary cortical-potential regulation tasks. Heartbeat distinguishes normal from abnormal heart sounds, represented by multivariate spectrogram features.
- **Motion trajectories.** Handwriting recognizes handwritten characters from pen-tip trajectories, whereas UWaveGestureLibrary classifies gestures from motion-sensor recordings.
- **Speech recordings.** JapaneseVowels identifies speakers from vowel utterances, while SpokenArabicDigits recognizes spoken Arabic digits from sequences of Mel-frequency cepstral coefficients.
- **Spectral measurements.** EthanolConcentration distinguishes four ethanol concentration levels from multivariate spectroscopic measurements.
- **Traffic observations.** PEMS-SF classifies the day of the week from traffic occupancy measurements collected by freeway sensors in the San Francisco Bay Area.

Data Splits and Preprocessing. For all datasets, we use the predefined training and test splits provided by the UEA archive, without randomly resampling or merging instances across the two splits. All preprocessing statistics are estimated exclusively from the training split and subsequently applied to the test split. Specifically, each variable is standardized using its training-set mean and standard deviation. Non-finite observations, when present, are replaced through linear interpolation within the corresponding time series. We preserve the original temporal order, sampling resolution, and multivariate channel structure of each dataset.

B Task Protocols and Implementation Details

B.1 Forecasting Task

Task Definition. Given a multivariate historical sequence $\mathbf{X} \in \mathbb{R}^{L_{\text{in}} \times C}$, where C denotes the number of variables, the forecasting task aims to predict the subsequent sequence $\mathbf{Y} \in \mathbb{R}^{L_{\text{out}} \times C}$. We fix the look-back length to $L_{\text{in}} = 96$ and evaluate the model over prediction horizons $L_{\text{out}} \in \{96, 192, 336, 720\}$.

Table 3: Statistics of the forecasting benchmarks. Length denotes the number of time steps in each benchmark file before splitting.

Dataset	Variables	Length	Frequency	Domain
ETTm1	7	69,680	15 minutes	Electricity
ETTm2	7	69,680	15 minutes	Electricity
ETTh1	7	17,420	1 hour	Electricity
ETTh2	7	17,420	1 hour	Electricity
Traffic	862	17,544	1 hour	Transportation
ECL	321	26,304	1 hour	Electricity
Weather	21	52,696	10 minutes	Weather

Table 4: Statistics of the ten UEA classification datasets. Length refers to the sequence length reported in the UEA 2018 archive.

Dataset	Train	Test	Variables	Length	Classes
EthanolConcentration	261	263	3	1,751	4
FaceDetection	5,890	3,524	144	62	2
Handwriting	150	850	3	152	26
Heartbeat	204	205	61	405	2
JapaneseVowels	270	370	12	29	9
PEMS-SF	267	173	963	144	7
SelfRegulationSCP1	268	293	6	896	2
SelfRegulationSCP2	200	180	7	1,152	2
SpokenArabicDigits	6,599	2,199	13	93	10
UWaveGestureLibrary	120	320	3	315	8

Evaluation Metrics. We evaluate forecasting performance using mean squared error (MSE) and mean absolute error (MAE), both averaged over test samples, prediction steps, and variables:

$$\mathrm{MSE} = \frac{1}{NL_{\mathrm{out}}C} \sum_{n=1}^{N} \sum_{t=1}^{L_{\mathrm{out}}} \sum_{c=1}^{C} \left(\hat{Y}_{n,t,c} - Y_{n,t,c} \right)^2, \tag{14}$$

$$\mathrm{MAE} = \frac{1}{NL_{\mathrm{out}}C} \sum_{n=1}^{N} \sum_{t=1}^{L_{\mathrm{out}}} \sum_{c=1}^{C} \left| \hat{Y}_{n,t,c} - Y_{n,t,c} \right|. \tag{15}$$

Here, N denotes the number of test samples, and $\hat{\mathbf{Y}}$ denotes the model predictions. Lower MSE and MAE values indicate better forecasting performance. For each dataset, the reported *Avg* results are calculated as the arithmetic mean of the corresponding metric over the four prediction horizons.

B.2 Classification Task

Task Definition. Given a multivariate time-series instance $\mathbf{X} \in \mathbb{R}^{T \times C}$, where T and C denote the sequence length and number of variables, respectively, the classification task aims to predict a sequence-level label $y \in \{1, \ldots, K\}$, where K is the number of classes. Unlike forecasting, which estimates future observations, time series classification requires the model to identify discriminative temporal and cross-variable patterns within the observed sequence.

Classification Architecture. To adapt Time DeCode to classification, we replace its autoregressive next multi-code prediction objective with masked code prediction, which recovers masked code indices from the observed context, and converting its forecasting backbone into an encoder-based architecture. Given an input sequence, the encoder first produces a sequence of latent representations. These representations are subsequently aggregated into a sample-level feature vector and mapped to class probabilities.

$$\mathbf{H} = f_{\mathrm{enc}}(\mathbf{X}), \qquad \mathbf{z} = \mathrm{Aggregate}(\mathbf{H}), \qquad \mathbf{p} = \mathrm{softmax}\left(g_{\mathrm{cls}}(\mathbf{z})\right). \tag{16}$$

Here, f_{enc} denotes the time-series encoder, $\mathrm{Aggregate}(\cdot)$ combines the encoded representations into a fixed-dimensional feature vector, and g_{cls} denotes the classification head. The predicted class is

given by

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{\arg\max} \; p_k. \tag{17}$$

Training Objective. The encoder and classification head are jointly optimized using the cross-entropy loss:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{N_{\text{tr}}} \sum_{n=1}^{N_{\text{tr}}} \sum_{k=1}^{K} \mathbb{I}[y_n = k] \log p_{n,k}, \tag{18}$$

where N_{tr} denotes the number of training instances and $p_{n,k}$ is the predicted probability that the n-th instance belongs to class k.

Evaluation Metric. We report classification accuracy on each dataset:

$$\text{Accuracy} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[\hat{y}_n = y_n] \times 100\%, \tag{19}$$

where N denotes the number of test instances and $\mathbb{I}[\cdot]$ is the indicator function. The overall classification result is calculated as the unweighted mean accuracy across the ten datasets, assigning equal importance to each dataset regardless of its test-set size.

B.3 Implementation Details

Our implementation uses PyTorch 2.7.1 with CUDA 12.6. The main forecasting and classification results of Time DeCode were obtained on NVIDIA H20 GPUs. A subset of the hyperparameter sensitivity and ablation experiments was conducted on NVIDIA GeForce RTX 4090 and RTX 5090 GPUs. Within each controlled comparison, all compared configurations were evaluated on the same GPU type, and only the component or hyperparameter under investigation was varied while the remaining experimental settings were kept fixed to ensure comparability.

C Full Results of the Main Experiments

C.1 Time Series Forecasting Results

This section provides detailed experimental results to complement the comparisons in Table 1.

Table 5 reports MSE and MAE for each of the four prediction horizons on all seven forecasting datasets. The input length is fixed to 96. Rankings are determined before rounding, and the first- and second-place counts exclude the Avg rows. Overall, Time DeCode achieves the best results in 51 of the 56 dataset–horizon–metric comparisons. Averaged over prediction horizons, it obtains the lowest MAE on all seven datasets and the lowest MSE on five. Compared with TimeFilter, the strongest baseline on average, Time DeCode reduces the overall MSE and MAE by 3.8% and 3.5%, respectively, demonstrating its strong forecasting performance across the evaluated benchmarks.

C.2 Time Series Classification Results

Table 6 presents the complete results on ten UEA classification datasets. Time DeCode achieves the highest mean accuracy of 74.18%, exceeding TimesNet, the strongest baseline on average, by 0.44 percentage points. It ranks first on eight datasets, including ties, and second on one other dataset. These results complement the forecasting evaluation and support the applicability of Time DeCode to classification.

D Detailed Denoising Experiments

D.1 Experimental Protocol

To provide a comprehensive evaluation of the denoising capability of Time DeCode, we study four corruption settings according to when and where perturbations enter the forecasting pipeline: testing stage corruption, training stage corruption, full stage corruption and inference stage corruption.

Table 5: Detailed long-term forecasting results with an input length of 96 and prediction horizons of 96, 192, 336, and 720. The best result is shown in bold, and the second-best result is underlined.

Dataset	Pred.	Time DeCode		Chronos-2 (2025)		TimeSiam (2024)		TOTEM (2024)		TimeFilter (2025)		iTransformer (2024)		PatchTST (2023)		TimesNet (2023)		DLinear (2023)		FEDformer (2022)		Informer (2021)		Autoformer (2021)	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	96	**0.311**	**0.344**	0.313	0.346	0.320	0.361	0.319	0.349	0.322	0.358	0.339	0.375	0.324	0.366	0.336	0.377	0.346	0.374	0.382	0.425	0.658	0.573	0.442	0.452
	192	**0.353**	**0.368**	0.355	0.369	0.357	0.382	0.372	0.381	0.365	0.384	0.381	0.396	0.367	0.389	0.388	0.402	0.382	0.391	0.419	0.443	0.732	0.618	0.573	0.515
	336	**0.386**	**0.394**	0.400	0.394	0.387	0.404	0.403	0.403	0.393	0.404	0.417	0.420	0.401	0.412	0.417	0.423	0.415	0.415	0.446	0.457	1.007	0.782	0.523	0.508
	720	**0.439**	**0.427**	0.500	0.446	0.443	0.438	0.478	0.442	0.451	0.440	0.493	0.461	0.457	0.442	0.483	0.460	0.473	0.451	0.516	0.491	0.930	0.707	0.538	0.517
ETTm2	96	**0.168**	**0.246**	0.175	0.247	0.176	0.261	0.178	0.254	0.171	0.256	0.183	0.269	0.177	0.261	0.188	0.268	0.193	0.293	0.191	0.280	0.501	0.570	0.220	0.302
	192	**0.240**	**0.291**	0.240	0.292	0.241	0.303	0.248	0.303	0.240	0.302	0.252	0.312	0.243	0.305	0.248	0.305	0.285	0.361	0.264	0.324	0.655	0.610	0.284	0.340
	336	**0.297**	**0.327**	0.313	0.339	0.301	0.341	0.331	0.358	0.301	0.341	0.315	0.351	0.305	0.346	0.316	0.345	0.385	0.430	0.326	0.361	1.318	0.865	0.329	0.364
	720	**0.401**	**0.394**	0.409	0.394	0.402	0.398	0.445	0.435	0.408	0.403	0.413	0.408	0.405	0.404	0.423	0.407	0.556	0.524	0.418	0.414	4.296	1.590	0.447	0.440
ETTh1	96	**0.362**	**0.388**	0.406	0.406	0.378	0.401	0.385	0.396	0.393	0.399	0.392	0.408	0.381	0.400	0.389	0.412	0.396	0.411	0.383	0.423	0.834	0.686	0.465	0.473
	192	**0.402**	**0.412**	0.451	0.438	0.423	0.429	0.447	0.438	0.439	0.425	0.449	0.443	0.432	0.435	0.439	0.442	0.445	0.440	0.430	0.450	0.961	0.756	0.488	0.482
	336	**0.437**	**0.432**	0.522	0.474	0.456	0.449	0.497	0.458	0.479	0.443	0.493	0.467	0.475	0.461	0.495	0.471	0.487	0.465	0.480	0.476	1.091	0.824	0.512	0.503
	720	**0.435**	**0.444**	0.726	0.605	0.460	0.467	0.568	0.512	0.502	0.479	0.513	0.500	0.521	0.503	0.520	0.495	0.513	0.510	0.556	0.529	1.195	0.882	0.565	0.545
ETTh2	96	**0.285**	**0.332**	0.297	0.337	0.290	0.342	0.297	0.340	0.299	0.351	0.300	0.351	0.285	0.339	0.330	0.370	0.341	0.395	0.339	0.382	3.060	1.387	0.418	0.440
	192	**0.362**	**0.381**	0.382	0.390	0.375	0.397	0.378	0.392	0.388	0.405	0.381	0.400	0.372	0.393	0.395	0.409	0.482	0.479	0.426	0.436	6.444	2.080	0.431	0.439
	336	**0.405**	**0.413**	0.406	0.416	0.413	0.427	0.426	0.432	0.432	0.439	0.431	0.436	0.422	0.434	0.471	0.468	0.593	0.542	0.463	0.471	4.538	1.764	0.476	0.491
	720	**0.422**	**0.435**	0.463	0.466	0.424	0.445	0.642	0.580	0.445	0.454	0.428	0.446	0.430	0.450	0.444	0.454	0.840	0.661	0.485	0.490	4.335	1.804	0.494	0.507
Traffic	96	**0.387**	**0.255**	0.483	0.302	0.429	0.278	0.523	0.302	0.390	0.260	0.393	0.268	0.545	0.360	0.598	0.316	0.668	0.411	0.575	0.356	0.713	0.405	0.619	0.384
	192	**0.409**	**0.264**	0.471	0.292	0.443	0.282	0.526	0.301	0.413	0.269	0.412	0.277	0.536	0.354	0.627	0.325	0.617	0.387	0.623	0.388	0.732	0.415	0.601	0.366
	336	0.426	**0.272**	0.486	0.296	0.457	0.288	0.549	0.308	0.429	0.276	**0.424**	0.283	0.546	0.355	0.636	0.337	0.623	0.390	0.611	0.378	0.796	0.445	0.661	0.410
	720	**0.458**	**0.292**	0.532	0.325	0.488	0.306	0.587	0.329	0.462	0.295	0.460	0.301	0.585	0.375	0.656	0.349	0.664	0.410	0.687	0.432	0.868	0.483	0.670	0.418
ECL	96	**0.138**	**0.230**	0.176	0.260	0.159	0.245	0.176	0.260	0.138	0.237	0.149	0.240	0.193	0.284	0.168	0.272	0.210	0.301	0.193	0.310	0.340	0.424	0.190	0.304
	192	**0.161**	**0.251**	0.185	0.269	0.168	0.255	0.187	0.272	0.165	0.260	0.165	0.256	0.198	0.289	0.185	0.287	0.210	0.305	0.214	0.330	0.380	0.458	0.231	0.340
	336	**0.167**	**0.260**	0.201	0.285	0.186	0.273	0.198	0.284	0.167	0.266	0.178	0.271	0.214	0.304	0.208	0.308	0.223	0.319	0.217	0.331	0.376	0.455	0.290	0.382
	720	0.204	0.295	0.242	0.319	0.229	0.310	0.236	0.318	**0.196**	**0.290**	0.228	0.313	0.255	0.337	0.265	0.350	0.258	0.350	0.315	0.416	0.491	0.522	0.309	0.406
Weather	96	**0.156**	**0.198**	0.178	0.201	0.174	0.217	0.163	0.207	0.162	0.210	0.182	0.223	0.179	0.219	0.172	0.222	0.196	0.256	0.261	0.339	0.409	0.452	0.241	0.315
	192	**0.205**	**0.242**	0.227	0.247	0.221	0.257	0.208	0.249	0.209	0.253	0.230	0.262	0.225	0.259	0.224	0.264	0.239	0.299	0.366	0.423	0.438	0.461	0.277	0.340
	336	0.260	**0.284**	0.277	0.286	0.274	0.294	**0.257**	0.288	0.265	0.293	0.284	0.301	0.279	0.298	0.284	0.305	0.281	0.331	0.389	0.430	0.611	0.554	0.341	0.383
	720	0.337	**0.332**	0.359	0.340	0.350	0.343	**0.329**	0.337	0.343	0.343	0.362	0.351	0.355	0.348	0.355	0.351	0.345	0.382	0.415	0.433	1.372	0.852	0.435	0.443
First Count		51		0		0		2		2		1		0		0		0		0		0		0	
Second Count		5		16		12		3		16		2		2		0		0		0		0		0	

Table 6: Classification accuracy (%) on ten UEA datasets. The best and second-best results are highlighted in bold and underlined text, respectively. Avg denotes the unweighted mean accuracy across datasets. First and Second Counts exclude Avg; ties receive equal credit. TOTEM uses the UEA-pretrained variant.

Dataset	Time DeCode	Chronos-2	TOTEM	iTransformer	PatchTST	TimesNet	DLinear	Informer
EthanolConcentration	**32.32**	25.10	26.62	27.76	28.52	31.94	29.28	30.42
FaceDetection	67.57	67.59	60.27	66.37	**68.93**	67.45	68.22	67.91
Handwriting	**32.47**	3.88	12.59	27.18	26.59	**32.47**	23.06	31.88
Heartbeat	**81.46**	72.20	77.56	75.61	71.71	80.98	75.61	78.05
JapaneseVowels	**98.11**	68.38	95.41	97.57	95.95	97.84	96.49	97.03
PEMS-SF	87.86	76.88	87.28	**90.17**	80.35	87.86	80.92	82.66
SelfRegulationSCP1	**92.15**	82.59	88.74	**92.15**	83.96	91.47	91.47	89.08
SelfRegulationSCP2	**60.56**	52.78	54.44	54.44	52.22	60.00	54.44	54.44
SpokenArabicDigits	**99.27**	96.32	96.77	98.04	97.64	**99.27**	96.50	98.73
UWaveGestureLibrary	**90.00**	84.06	80.31	85.94	85.31	88.13	82.19	86.56
Avg	**74.18**	62.98	68.00	71.52	69.12	73.74	69.82	71.68
First Count	8	0	0	2	1	2	0	0
Second Count	1	0	0	0	0	7	2	2

For testing stage corruption, we perturb only the historical inputs provided to trained models, while keeping the model checkpoints and forecasting targets unchanged. This setting evaluates whether the models can make accurate predictions from noisy or incomplete observations without corruption-specific retraining. We consider five categories of corruption: additive noise, outliers, random missingness, block missingness, and level shifts.

For training stage corruption, Gaussian noise, outliers, or random missingness is introduced only into the temporal contexts used during pretraining, while the corresponding learning targets remain clean. The resulting models are subsequently fine-tuned and evaluated on clean data. This setting examines whether the pretraining procedure can learn predictive temporal representations from corrupted contexts.

We further consider corruption throughout training, where historical inputs are perturbed during every applicable training stage, including pretraining and downstream fine-tuning. The corresponding learning targets, validation data, and test data remain clean. Compared with training stage corruption, this setting provides a more challenging evaluation in which clean historical inputs are unavailable throughout the complete training process.

Finally, for inference-stage corruption, one or more predicted codes are replaced at an early autoregressive feedback step before being incorporated into the context used for subsequent predictions. This setting directly evaluates whether errors in intermediate discrete representations accumulate during forecasting and whether code-space prediction calibration improves inference stability.

Unless otherwise specified, all experiments are conducted on ETTh1 with both the input length and prediction horizon fixed to 96. MSE is used as the primary evaluation metric. The following subsections describe the corruption mechanisms, comparison methods, severity levels, and experimental results for each setting.

D.2 Testing Stage Corruption

Experimental Setting. For testing stage corruption, noise is introduced only at the testing stage and is applied exclusively to the historical inputs of the test set. The forecasting targets and model parameters remain unchanged, and no corruption-specific retraining is performed. This setting evaluates whether each method can maintain accurate forecasts when the observed context contains noise, missing values, outliers, or local distribution shifts.

We consider eight types of corruption, covering additive noise, sparse anomalies, incomplete observations, and local distribution shifts. Let $\mathbf{x} \in \mathbb{R}^{T\times C}$ denote a standardized historical input containing T time steps and C variables. All corruptions are applied only to $\mathbf{x}$ before RevIN, while the forecasting targets remain unchanged. The corruption types are defined as follows:

- **Gaussian noise.** We add independent zero-mean Gaussian noise to every input entry:

$$\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \qquad \epsilon_{t,c} \sim \mathcal{N}(0, \sigma^2). \tag{20}$$

  We evaluate $\sigma \in \{0.05, 0.10, 0.15\}$ in training-standardized units. This corruption represents independent measurement errors or small random fluctuations in the observed sequence.

- **Laplace noise.** We perturb each input entry with independent Laplace noise:

$$\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \qquad \epsilon_{t,c} \sim \text{Laplace}\left(0, \frac{\sigma}{\sqrt{2}}\right). \tag{21}$$

  The scale is normalized so that the standard deviation of the perturbation is σ. We use $\sigma \in \{0.05, 0.10, 0.15\}$. Compared with Gaussian noise, Laplace noise has heavier tails and therefore produces occasional larger deviations.

- **Student-t noise.** To evaluate robustness to more strongly heavy-tailed fluctuations, we sample noise from a Student-t distribution with three degrees of freedom:

$$\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \qquad \epsilon_{t,c} = \frac{\sigma}{\sqrt{3}} u_{t,c}, \quad u_{t,c} \sim t_3. \tag{22}$$

  Because a Student-t variable with three degrees of freedom has variance 3, the factor $1/\sqrt{3}$ normalizes its standard deviation to σ. We evaluate $\sigma \in \{0.05, 0.10, 0.15\}$.

- **Ornstein–Uhlenbeck noise.** We introduce temporally correlated noise independently for each variable. The perturbation is generated recursively as

$$\epsilon_{t,c} = \rho \epsilon_{t-1,c} + \sqrt{1-\rho^2}\, z_{t,c}, \qquad z_{t,c} \sim \mathcal{N}(0,1), \tag{23}$$

  where $\rho = 0.9$. The resulting sequence is multiplied by σ and added to the input, i.e., $\widetilde{\mathbf{x}} = \mathbf{x} + \sigma\boldsymbol{\epsilon}$. We use $\sigma \in \{0.05, 0.10, 0.15\}$. This setting simulates smooth disturbances that remain correlated over consecutive time steps.

- **Outliers.** We independently select each input entry as an outlier with probability r. A perturbation of either $+2$ or -2, sampled with equal probability, is then added to every selected entry:

$$\widetilde{x}_{t,c} = x_{t,c} + m_{t,c} s_{t,c}, \qquad m_{t,c} \sim \text{Bernoulli}(r), \quad s_{t,c} \in \{-2, +2\}. \tag{24}$$

  We evaluate outlier ratios $r \in \{0.1\%, 0.3\%, 0.5\%\}$. The perturbation magnitude is defined in training-standardized units. This corruption represents sparse anomalous observations rather than continuously distributed noise.

- **Random missingness.** Each input entry is independently marked as missing with probability r, where $r \in \{5\%, 10\%, 15\%\}$. A missing entry is replaced with the most recent available value of the same variable. If the first observation of a variable is missing, it is replaced with zero, which corresponds to the mean in the standardized input space. This setting simulates observations that are randomly unavailable across time and variables.
- **Block missingness.** For each variable, we randomly select a contiguous temporal segment with length approximately rT and treat all observations within that segment as missing, where $r \in \{5\%, 10\%, 15\%\}$. Missing entries are handled using the same forward-filling procedure as random missingness, with zero used when no preceding observation is available. This corruption represents temporary sensor failures or communication interruptions that remove consecutive observations.
- **Level shifts.** We randomly select a contiguous segment spanning 25% of the input window and add a constant offset to all observations within this segment:

$$\widetilde{x}_{t,c} = x_{t,c} + s_c\delta, \qquad s_c \in \{-1, +1\}, \tag{25}$$

  where the direction s_c is independently sampled for each variable. We evaluate shift magnitudes $\delta \in \{0.1, 0.2, 0.5\}$ in training-standardized units. This setting simulates abrupt local changes in the operating level or calibration of a temporal variable.

For every corruption type and severity level, we generate three independently corrupted test sets using perturbation seeds 42001, 42002, and 42003. Each reported severity-specific result is averaged over these three seeds. When reporting a single result for an entire corruption type, we further average over its three severity levels.

Results and Discussion. Tables 7 and 8 report the detailed results under different types and intensities of data-level corruption. Overall, Time DeCode achieves the lowest MSE in 23 of the 24 corrupted settings and the lowest MAE in all 24 settings. When averaged across all corruption types and severity levels, Time DeCode obtains an MSE of 0.3690 and an MAE of 0.3931. Compared with the strongest baseline for each metric, it reduces the average MSE by 3.83% relative to TimeSiam and the average MAE by 1.71% relative to TOTEM. These results indicate that Time DeCode can preserve temporal information useful for forecasting under additive noise, sparse anomalies, missing observations, and local distribution shifts.

As expected, the errors generally increase with the corruption severity, particularly under temporally correlated OU noise, block missingness, and large level shifts. Nevertheless, Time DeCode maintains a consistent advantage across most severity levels. The only exception in terms of MSE occurs under 15% block missingness, where TOTEM obtains an MSE of 0.4154 and Time DeCode ranks second with 0.4176. Time DeCode still achieves the best MAE in this setting. This exception is reasonable because long contiguous missing segments remove complete local temporal contexts rather than introducing fluctuations that can be directly filtered. Finally, the relatively small standard deviations across the three perturbation seeds indicate that the observed performance differences are generally stable and are not caused by a particular corruption realization.

D.3 Training Stage Corruption

Experimental Setting. For training stage corruption, noise is introduced only into the input contexts used during pretraining, while the corresponding prediction or reconstruction targets remain clean. After pretraining, all models are fine-tuned and evaluated on clean data. This setting evaluates whether each method can learn useful predictive representations from noisy, incomplete, or corrupted pretraining inputs.

All experiments are conducted on ETTh1 with both the input length and prediction horizon fixed to 96. We use a fixed training seed of 42 and a corruption seed of 42001. For each corruption type, we evaluate three severity levels and report the average MSE across these levels. No external pretraining data are used: Time DeCode and all pretraining-based baselines are trained using only the training split of ETTh1.

We consider the following three objective-level perturbations:

Table 7: Detailed MSE results under testing stage corruption on ETTh1 with both the input length and prediction horizon fixed to 96. Corrupted results are reported as the mean ± sample standard deviation over three perturbation seeds. The clean result is obtained from a single fixed evaluation. Bold and underlined values indicate the best and second-best mean results, respectively.

Category	Corruption	Severity	Time DeCode	TimeFilter	PatchTST	TOTEM	TimeSiam	Chronos-2
Control	Clean	–	**0.3615**	0.3928	0.3806	0.3845	0.3782	0.4064
Additive	Gaussian	0.05	**0.3626 ± 0.00026**	0.3932 ± 0.00007	0.3810 ± 0.00006	0.3847 ± 0.00009	0.3785 ± 0.00006	0.4062 ± 0.00006
		0.10	**0.3649 ± 0.00019**	0.3943 ± 0.00012	0.3821 ± 0.00014	0.3855 ± 0.00012	0.3793 ± 0.00012	0.4063 ± 0.00012
		0.15	**0.3686 ± 0.00029**	0.3963 ± 0.00019	0.3842 ± 0.00022	0.3867 ± 0.00011	0.3805 ± 0.00018	0.4069 ± 0.00017
Additive	Laplace	0.05	**0.3624 ± 0.00014**	0.3931 ± 0.00003	0.3810 ± 0.00007	0.3846 ± 0.00004	0.3785 ± 0.00004	0.4063 ± 0.00014
		0.10	**0.3643 ± 0.00009**	0.3943 ± 0.00007	0.3821 ± 0.00014	0.3854 ± 0.00013	0.3793 ± 0.00007	0.4064 ± 0.00022
		0.15	**0.3675 ± 0.00018**	0.3963 ± 0.00011	0.3842 ± 0.00020	0.3866 ± 0.00015	0.3806 ± 0.00011	0.4070 ± 0.00024
Additive	Student-t	0.05	**0.3624 ± 0.00004**	0.3931 ± 0.00005	0.3809 ± 0.00004	0.3847 ± 0.00013	0.3785 ± 0.00003	0.4063 ± 0.00008
		0.10	**0.3644 ± 0.00004**	0.3943 ± 0.00011	0.3821 ± 0.00009	0.3854 ± 0.00017	0.3792 ± 0.00007	0.4064 ± 0.00016
		0.15	**0.3676 ± 0.00009**	0.3963 ± 0.00017	0.3841 ± 0.00012	0.3866 ± 0.00026	0.3805 ± 0.00012	0.4071 ± 0.00023
Additive	OU	0.05	**0.3628 ± 0.00021**	0.3936 ± 0.00006	0.3815 ± 0.00014	0.3852 ± 0.00013	0.3790 ± 0.00009	0.4072 ± 0.00002
		0.10	**0.3663 ± 0.00035**	0.3960 ± 0.00011	0.3841 ± 0.00028	0.3874 ± 0.00008	0.3812 ± 0.00018	0.4100 ± 0.00010
		0.15	**0.3717 ± 0.00068**	0.4000 ± 0.00019	0.3885 ± 0.00039	0.3911 ± 0.00010	0.3848 ± 0.00028	0.4146 ± 0.00031
Sparse	Outliers	0.1%	**0.3622 ± 0.00021**	0.3934 ± 0.00008	0.3812 ± 0.00009	0.3848 ± 0.00005	0.3788 ± 0.00007	0.4065 ± 0.00001
		0.3%	**0.3638 ± 0.00019**	0.3948 ± 0.00010	0.3824 ± 0.00007	0.3856 ± 0.00010	0.3798 ± 0.00009	0.4070 ± 0.00009
		0.5%	**0.3654 ± 0.00010**	0.3961 ± 0.00012	0.3837 ± 0.00016	0.3864 ± 0.00008	0.3809 ± 0.00020	0.4075 ± 0.00011
Incomplete	Random missing	5%	**0.3635 ± 0.00022**	0.3939 ± 0.00004	0.3816 ± 0.00017	0.3851 ± 0.00024	0.3794 ± 0.00009	0.4072 ± 0.00003
		10%	**0.3656 ± 0.00024**	0.3956 ± 0.00019	0.3832 ± 0.00028	0.3863 ± 0.00031	0.3811 ± 0.00032	0.4086 ± 0.00007
		15%	**0.3687 ± 0.00023**	0.3979 ± 0.00022	0.3853 ± 0.00030	0.3882 ± 0.00041	0.3834 ± 0.00030	0.4106 ± 0.00022
Incomplete	Block missing	5%	**0.3673 ± 0.00029**	0.3985 ± 0.00050	0.3878 ± 0.00036	0.3899 ± 0.00041	0.3863 ± 0.00040	0.4132 ± 0.00047
		10%	**0.3892 ± 0.00140**	0.4119 ± 0.00160	0.4074 ± 0.00134	0.4050 ± 0.00190	0.4075 ± 0.00128	0.4367 ± 0.00243
		15%	0.4176 ± 0.00242	0.4219 ± 0.00054	0.4202 ± 0.00051	**0.4154 ± 0.00140**	0.4215 ± 0.00034	0.4617 ± 0.00208
Distributional	Level shift	0.1	**0.3625 ± 0.00020**	0.3931 ± 0.00005	0.3812 ± 0.00012	0.3848 ± 0.00014	0.3787 ± 0.00010	0.4071 ± 0.00013
		0.2	**0.3649 ± 0.00027**	0.3943 ± 0.00011	0.3832 ± 0.00026	0.3860 ± 0.00026	0.3802 ± 0.00022	0.4097 ± 0.00018
		0.5	**0.3802 ± 0.00113**	0.4035 ± 0.00041	0.3984 ± 0.00094	0.3952 ± 0.00054	0.3919 ± 0.00065	0.4296 ± 0.00028

Table 8: Detailed MAE results under testing stage corruption on ETTh1 with both the input length and prediction horizon fixed to 96. Corrupted results are reported as the mean ± sample standard deviation over three perturbation seeds. The clean result is obtained from a single fixed evaluation. Bold and underlined values indicate the best and second-best mean results, respectively.

Category	Corruption	Severity	Time DeCode	TimeFilter	PatchTST	TOTEM	TimeSiam	Chronos-2
Control	Clean	–	**0.3878**	0.3990	0.3995	0.3962	0.4009	0.4062
Additive	Gaussian	0.05	**0.3884 ± 0.00005**	0.3994 ± 0.00004	0.3999 ± 0.00004	0.3964 ± 0.00002	0.4012 ± 0.00004	0.4064 ± 0.00005
		0.10	**0.3901 ± 0.00007**	0.4008 ± 0.00007	0.4013 ± 0.00010	0.3974 ± 0.00002	0.4021 ± 0.00008	0.4072 ± 0.00009
		0.15	**0.3931 ± 0.00012**	0.4030 ± 0.00011	0.4036 ± 0.00015	0.3989 ± 0.00004	0.4036 ± 0.00013	0.4084 ± 0.00012
Additive	Laplace	0.05	**0.3884 ± 0.00003**	0.3994 ± 0.00003	0.4000 ± 0.00002	0.3964 ± 0.00003	0.4012 ± 0.00001	0.4064 ± 0.00009
		0.10	**0.3900 ± 0.00011**	0.4008 ± 0.00005	0.4013 ± 0.00003	0.3973 ± 0.00006	0.4021 ± 0.00001	0.4071 ± 0.00015
		0.15	**0.3928 ± 0.00006**	0.4030 ± 0.00005	0.4037 ± 0.00004	0.3989 ± 0.00008	0.4037 ± 0.00001	0.4083 ± 0.00017
Additive	Student-t	0.05	**0.3884 ± 0.00005**	0.3994 ± 0.00006	0.3999 ± 0.00004	0.3964 ± 0.00009	0.4012 ± 0.00005	0.4064 ± 0.00005
		0.10	**0.3899 ± 0.00014**	0.4007 ± 0.00011	0.4012 ± 0.00006	0.3973 ± 0.00016	0.4021 ± 0.00010	0.4071 ± 0.00009
		0.15	**0.3926 ± 0.00017**	0.4028 ± 0.00014	0.4034 ± 0.00007	0.3987 ± 0.00017	0.4036 ± 0.00015	0.4082 ± 0.00011
Additive	OU	0.05	**0.3891 ± 0.00018**	0.3999 ± 0.00006	0.4005 ± 0.00008	0.3971 ± 0.00009	0.4017 ± 0.00012	0.4071 ± 0.00004
		0.10	**0.3927 ± 0.00019**	0.4026 ± 0.00012	0.4033 ± 0.00017	0.3996 ± 0.00011	0.4040 ± 0.00021	0.4099 ± 0.00011
		0.15	**0.3984 ± 0.00051**	0.4071 ± 0.00018	0.4079 ± 0.00023	0.4038 ± 0.00012	0.4077 ± 0.00030	0.4145 ± 0.00025
Sparse	Outliers	0.1%	**0.3887 ± 0.00016**	0.3997 ± 0.00007	0.4001 ± 0.00009	0.3966 ± 0.00006	0.4015 ± 0.00008	0.4065 ± 0.00002
		0.3%	**0.3906 ± 0.00007**	0.4010 ± 0.00008	0.4014 ± 0.00006	0.3974 ± 0.00005	0.4026 ± 0.00007	0.4070 ± 0.00004
		0.5%	**0.3925 ± 0.00023**	0.4022 ± 0.00014	0.4026 ± 0.00014	0.3983 ± 0.00017	0.4037 ± 0.00016	0.4076 ± 0.00004
Incomplete	Random missing	5%	**0.3890 ± 0.00012**	0.4000 ± 0.00004	0.4003 ± 0.00008	0.3968 ± 0.00012	0.4018 ± 0.00002	0.4068 ± 0.00004
		10%	**0.3903 ± 0.00014**	0.4014 ± 0.00006	0.4015 ± 0.00013	0.3979 ± 0.00011	0.4030 ± 0.00016	0.4078 ± 0.00005
		15%	**0.3921 ± 0.00015**	0.4032 ± 0.00007	0.4029 ± 0.00012	0.3993 ± 0.00016	0.4046 ± 0.00015	0.4091 ± 0.00010
Incomplete	Block missing	5%	**0.3914 ± 0.00002**	0.4035 ± 0.00015	0.4044 ± 0.00006	0.4002 ± 0.00013	0.4066 ± 0.00006	0.4107 ± 0.00015
		10%	**0.4034 ± 0.00064**	0.4135 ± 0.00067	0.4178 ± 0.00046	0.4114 ± 0.00075	0.4206 ± 0.00053	0.4255 ± 0.00077
		15%	**0.4171 ± 0.00132**	0.4205 ± 0.00021	0.4260 ± 0.00042	0.4193 ± 0.00030	0.4291 ± 0.00020	0.4398 ± 0.00072
Distributional	Level shift	0.1	**0.3887 ± 0.00018**	0.3994 ± 0.00011	0.4002 ± 0.00013	0.3967 ± 0.00010	0.4014 ± 0.00009	0.4069 ± 0.00014
		0.2	**0.3913 ± 0.00034**	0.4008 ± 0.00024	0.4024 ± 0.00024	0.3981 ± 0.00021	0.4030 ± 0.00019	0.4093 ± 0.00025
		0.5	**0.4060 ± 0.00071**	0.4106 ± 0.00078	0.4178 ± 0.00085	0.4083 ± 0.00052	0.4152 ± 0.00064	0.4276 ± 0.00039

- **Gaussian noise.** Independent Gaussian noise is added to every observed entry in the pretraining input:

$$\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \qquad \epsilon_{t,c} \sim \mathcal{N}(0, \sigma^2), \tag{26}$$

where $\sigma \in \{0.05, 0.10, 0.15\}$. The corresponding learning targets remain clean.

Table 9: Detailed forecasting results under training stage perturbations on ETTh1, with both the input length and prediction horizon fixed to 96. All experiments use a training seed of 42. Bold and underlined values indicate the best and second-best results, respectively. Lower is better.

Corruption	Severity	Time DeCode		TimeSiam		PatchTST Recon.		TOTEM		Chronos-2	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Gaussian	0.05	**0.3626**	**0.3878**	<u>0.3779</u>	0.4007	0.3814	0.4016	0.3840	<u>0.3956</u>	0.3879	0.4005
	0.10	**0.3616**	**0.3864**	<u>0.3778</u>	0.4007	0.3811	0.4017	0.3902	<u>0.3982</u>	0.3876	0.4003
	0.15	**0.3662**	**0.3901**	<u>0.3779</u>	0.4009	0.3809	0.4021	0.3825	<u>0.3960</u>	0.3924	0.4041
Outliers	0.1%	**0.3631**	**0.3884**	<u>0.3782</u>	0.4009	0.3810	0.4017	0.3833	<u>0.3953</u>	0.4040	0.4065
	0.3%	**0.3625**	**0.3881**	<u>0.3782</u>	0.4009	0.3810	0.4017	0.3812	<u>0.3936</u>	0.4047	0.4065
	0.5%	**0.3633**	**0.3886**	<u>0.3782</u>	0.4009	0.3810	0.4017	0.3824	<u>0.3943</u>	0.4063	0.4079
Random missing	5%	**0.3645**	**0.3879**	<u>0.3779</u>	0.4007	0.3814	0.4016	0.3828	<u>0.3947</u>	0.3920	0.4021
	10%	**0.3657**	**0.3893**	<u>0.3778</u>	0.4007	0.3811	0.4017	0.3873	<u>0.4000</u>	0.4006	0.4042
	15%	**0.3629**	**0.3912**	<u>0.3779</u>	0.4009	0.3809	0.4021	0.3859	<u>0.3979</u>	0.4003	0.4040

- **Outliers.** We independently select input entries with probability r and add either $+2$ or -2 in training-standardized units:

$$\widetilde{x}_{t,c} = x_{t,c} + m_{t,c}s_{t,c}, \qquad m_{t,c} \sim \text{Bernoulli}(r), \quad s_{t,c} \in \{-2, +2\}. \tag{27}$$

  We evaluate $r \in \{0.1\%, 0.3\%, 0.5\%\}$. This setting introduces sparse, high-magnitude disturbances into the pretraining context without modifying its target.

- **Random missingness.** Each input entry is independently masked with probability $r \in \{5\%, 10\%, 15\%\}$. Masked entries are replaced with zero in the standardized input space, while the clean observations are retained in the learning target. This setting evaluates whether the pretraining objective can recover predictive temporal information from incomplete observations.

We compare Time DeCode with four pretraining-based methods. For all methods, perturbations are introduced only during the pretraining stage, while downstream fine-tuning and evaluation are performed on clean data. The learning targets remain clean throughout pretraining, resulting in a consistent noisy-input–clean-target protocol across the compared methods.

Results and Discussion. As shown in Table 2, Time DeCode achieves the lowest MSE under all three training stage corruption categories. Its averaged MSE values are 0.3635 under Gaussian noise, 0.3630 under outlier corruption, and 0.3643 under random missingness. Averaged across the three corruption categories, Time DeCode obtains an MSE of 0.3636, reducing the error of TimeSiam, the strongest baseline in this experiment, by 3.81%. Time DeCode also consistently outperforms PatchTST reconstruction pretraining, TOTEM, and Chronos-2.

The advantage over reconstruction-based pretraining suggests that directly recovering continuous input values may retain local disturbances that are not useful for future prediction. In contrast, the token-prediction objective of Time DeCode encourages the model to recover stable discrete temporal patterns from corrupted observations. The similar performance of Time DeCode across Gaussian noise, sparse outliers, and random missingness further indicates that its improvement is not restricted to a particular noise distribution. Overall, these results provide empirical support for the proposed objective-level denoising design: Time DeCode can learn predictive temporal representations from noisy or incomplete pretraining inputs while maintaining strong performance on clean downstream forecasting data.

D.4 Full Stage Corruption

We consider a more challenging setting in which the training data remain corrupted throughout all applicable training stages. In this setting, noise is introduced into the historical inputs during every applicable training stage, including pretraining and downstream fine-tuning. The corresponding learning targets, validation data, and test data remain clean. Compared with pretraining-stage corruption, this setting provides a more challenging evaluation in which clean historical inputs are unavailable throughout the training process.

Table 10: Forecasting MSE on ETTh1 when historical inputs are corrupted throughout all applicable training stages. Both the input length and prediction horizon are 96. Avg. denotes the average over the three severity levels, and Overall Avg. denotes the average over all nine corrupted settings. Bold and underlined values indicate the best and second-best results in each row, respectively.

Corruption	Severity	Time DeCode	TimeFilter	PatchTST	TOTEM	TimeSiam
Gaussian noise	0.05	**0.3637**	0.3865	0.3815	0.3826	0.3787
	0.10	**0.3630**	0.3832	0.3815	0.3820	0.3791
	0.15	**0.3664**	0.3816	0.3818	0.3868	0.3797
	Avg.	**0.3644**	0.3838	0.3816	0.3838	0.3792
Outliers	0.1%	**0.3643**	0.3890	0.3807	0.3835	0.3788
	0.3%	**0.3612**	0.3869	0.3815	0.3865	0.3790
	0.5%	**0.3646**	0.3926	0.3820	0.3832	0.3796
	Avg.	**0.3634**	0.3895	0.3814	0.3844	0.3791
Random missingness	5%	**0.3675**	0.3834	0.3816	0.3845	0.3796
	10%	**0.3672**	0.3917	0.3836	0.3850	0.3816
	15%	**0.3662**	0.3947	0.3870	0.3959	0.3842
	Avg.	**0.3670**	0.3899	0.3841	0.3885	0.3818
Overall Avg.		**0.3649**	0.3877	0.3824	0.3856	0.3800

We conduct experiments on ETTh1 with both the input length and prediction horizon fixed to 96. We consider Gaussian noise with standard deviations $\{0.05, 0.10, 0.15\}$, sparse outliers affecting $\{0.1\%, 0.3\%, 0.5\%\}$ of the input entries with magnitude ± 2 in standardized units, and random missingness with ratios $\{5\%, 10\%, 15\%\}$, where missing entries are replaced by zero in standardized space. All experiments use training seed 42 and corruption seed 42001.

As shown in Table 10, Time DeCode achieves the lowest MSE in all nine corruption–severity settings. Averaged over all settings, it obtains an MSE of 0.3649, outperforming TimeSiam, the strongest baseline, by 3.98%. The improvements over TimeSiam are 3.90%, 4.16%, and 3.88% under Gaussian noise, outliers, and random missingness, respectively. These results show that the advantage of Time DeCode is maintained when noisy observations affect the complete training pipeline instead of only a single pretraining stage. This finding provides additional evidence that its noise-aware discrete representation can retain forecasting-relevant temporal information under persistent corruption of the training inputs.

D.5 Inference Stage Corruption

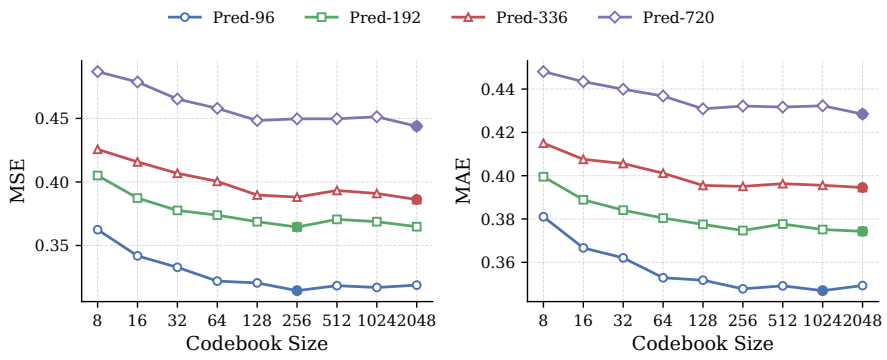
Experimental Setting. To evaluate the stability of autoregressive inference, we introduce errors into the predicted codes that are fed back into the model for subsequent prediction. At the first feedback step, we randomly replace one, two, or four predicted codes for each sample and variable. Each selected code is replaced with a different code sampled uniformly from the remaining entries in the codebook. Predictions produced before the replacement remain unchanged; therefore, the perturbation affects forecasting only after the corrupted codes are incorporated into the subsequent context.

We conduct the experiment on ETTh1 with both the input length and prediction horizon fixed to 96. The forecasting checkpoint is trained with seed 42 and remains fixed throughout the evaluation. For each replacement strength, we independently generate the code replacements using three perturbation seeds, i.e., $\{42001, 42002, 42003\}$, and report the mean and standard deviation of MSE and MAE. We compare the standard inference procedure using code-space prediction calibration with a non-overlapping variant denoted as w/o CPC. All other model parameters and evaluation settings are held fixed.

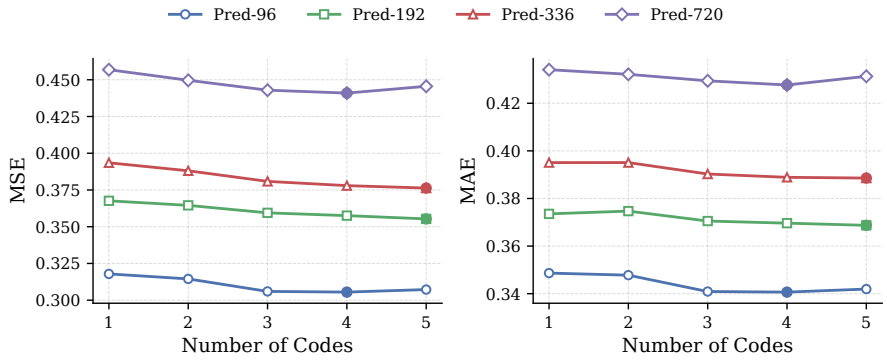
Results and Discussion. As shown in Table 11, CPC consistently achieves lower MSE and MAE than its non-overlapping counterpart at all three perturbation levels. As the number of replaced codes increases from one to four, the forecasting errors generally increase, indicating that stronger perturbations to the feedback representations have a greater influence on subsequent predictions.

Table 11: Forecasting performance under inference-level code replacement on ETTh1. One, two, or four predicted codes are replaced at the first feedback step. Results report the mean and standard deviation over three perturbation seeds. Improvement denotes the relative error reduction of CPC over w/o CPC. Lower is better.

Replaced codes	Metric	CPC	w/o CPC	Improvement
1	MSE	**0.361425±0.000078**	0.361597±0.000029	**0.048%**
	MAE	**0.387869±0.000022**	0.388073±0.000016	**0.052%**
2	MSE	**0.361723±0.000187**	0.361918±0.000121	**0.054%**
	MAE	**0.388167±0.000039**	0.388396±0.000038	**0.059%**
4	MSE	**0.361923±0.000265**	0.362126±0.000235	**0.056%**
	MAE	**0.388547±0.000060**	0.388806±0.000043	**0.067%**



(a) Codebook size sensitivity on ETTm1.



(b) Sensitivity to the number of residual codes on ETTm1.

Figure 7: Parameter sensitivity analysis of Time DeCode. Panels (a) and (b) report MSE and MAE under four prediction horizons on ETTm1. Vertical scales differ across plots.

E Parameter Study

To further examine how the degree of discretization affects forecasting performance, we study the codebook size and the number of residual codes on ETTm1 under four prediction horizons, $\{96, 192, 336, 720\}$.

Effect of Codebook Size. We vary the codebook size from 8 to 2048. As shown in Figure 7a, increasing the size of a small codebook generally improves forecasting performance across the four prediction horizons, suggesting that sufficient codebook capacity is necessary to represent diverse temporal patterns. Beyond moderate sizes, the gains become smaller and non-monotonic. For MSE, a codebook size of 256 performs best at horizons 96 and 192, whereas 2048 gives the best results

at horizons 336 and 720. These findings suggest that longer-horizon forecasting can benefit from greater representational capacity, although enlarging the codebook does not consistently improve performance.

Effect of the Number of Codes. We vary the number of residual codes from one to five. Figure 7b shows that increasing the number from one to four reduces MSE across all four prediction horizons, supporting the use of multiple residual codes to refine discrete representations. Adding a fifth code further improves MSE at horizons 192 and 336 but slightly worsens it at horizons 96 and 720. The MAE results similarly show diminishing gains as more codes are introduced. Overall, multiple residual codes improve representation capacity, while the benefit of additional codes depends on the forecasting setting.