INVESTIGATING THE EFFECTS OF PATCH SIZE IN TIME SERIES FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Time series analysis is a core task across science, industry, and decision-making, and transformer-based foundation models have recently emerged as a promising unified approach. A critical step in designing transformer-based Time Series Foundation Models (TSFMs) is tokenization. Most existing TSFMs use patch-based tokenization, making the patch size a central design choice that controls temporal resolution, token sequence length, and computational cost. In this work, we systematically study the role of patch size and architecture choice in time series foundation models, two decisions that are typically studied independently but, as we show, are deeply coupled. We compare encoder-only and decoder-only forecasting models trained on the same data mixture and with the same optimization budget, varying the patch size within each model class. The two architectures respond differently: across 97 GIFT-Eval tasks, encoder-only models improve as patch sizes decrease, whereas decoder-only models perform best at an intermediate patch size, reflecting a dataset-level preference that shifts towards larger patches as the forecasting horizon grows. On UCR-91 classification with frozen representations, smaller patches again help the encoder, while the decoder is hardly affected. Encoder-only models also benefit more from longer context windows, though gains saturate beyond 2048 context length. The two architectures occupy complementary positions on the accuracy–compute trade-off: decoder-only models are more efficient, while encoder-only models achieve higher accuracy. These findings suggest that patch size and architecture should not be chosen in isolation, but should both follow from the target forecasting horizon and deployment constraints.

1 INTRODUCTION

Recent years have witnessed an increasing interest in Time Series Foundation Models (TSFMs), which promise to replace dataset- and task-specific pipelines with pretrained models capable of transferring across domains, sampling frequencies, context lengths, prediction horizons, and, in some cases, across time series tasks.

Despite rapid empirical progress, the field has not converged towards a canonical architectural recipe. The dominant family is patch-based transformers (Vaswani et al., 2017), within which two designs coexist: PatchTST-FM (Wen et al., 2026) and Chronos-2 (Ansari et al., 2025) use encoder-only, full-attention backbones to predict masked future patches in parallel, while TimesFM (Das et al., 2024), Moirai 2.0 (Liu et al., 2025), and Toto (Cohen et al., 2025) employ causal decoder-only layers for single or multiple next-patch forecasting. Competitive models are also built on other backbones, such as the recurrent xLSTM layers of the TiRex family (Auer et al., 2025) or the state-space architecture of FlowState (Graf et al., 2026). Benchmark performance alone has not identified a preferred architecture, and the identification of critical components is further obfuscated as published models differ simultaneously in pretraining corpus, scale, training objective, and tokenization. Since these choices move together, the contribution of any single one cannot be isolated from reported results.

What transformer-based models share, across architecture types, is the need to map raw observations to tokens, most commonly by segmenting the series into patches. Inspired by the patch-based tokenization of Vision Transformers (Dosovitskiy et al., 2021), PatchTST (Nie et al., 2023) popularized this approach for time series by segmenting each univariate series into fixed-size patches that are

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.

embedded as input tokens. Patch size controls temporal granularity and computational cost at both training and inference time. Some models adopt multiple temporal scales: Moirai (Woo et al., 2024) uses different patch sizes across sampling frequencies, while Kairos (Feng et al., 2025) and TimeSqueeze (Ankireddy et al., 2025) adapt tokenization to the content of the series. However, recent evidence suggests that adaptive patching does not yet consistently improve performance over fixed patching (Zucchi et al., 2026).

The role of token granularity has also been investigated beyond time series. In vision, Wang et al. (2025) reports improved predictive performance with progressively smaller patches across several tasks and architectures. In diffusion transformers, increasing the number of input tokens improves generation quality but increases computation with little change in parameter count (Peebles and Xie, 2023). In language modeling, Limisiewicz et al. (2026) show that compute-optimal token compression depends on the training budget. These findings motivate a closer examination of *patch size* itself and its interaction with data and model architectures for time series.

Patch size has been ablated before, but always within a single model family, on aggregate forecasting error over benchmarks such as GIFT-Eval (Aksu et al., 2024), and on pretraining corpora that are not always fully public. Such ablations cannot tell whether a patch-size preference is a property of the architecture or of the particular training recipe it was measured with. Patch size has also been linked to the rank structure and compressibility of TSFM representations (Yu et al., 2026), suggesting that its effect extends beyond forecasting, but this analysis is again restricted to a single family. Three questions, therefore, remain open: whether encoder-only and decoder-only TSFMs prefer the same patch size once trained under identical conditions, whether an aggregate optimum hides a horizon-dependent preference at the dataset level, and whether the preferences observed in forecasting carry over to the representations these models learn.

We address these questions with a controlled study of patch size in encoder-only and decoder-only patch-based transformer TSFMs. We restrict the study to the transformer family, as it remains dominant, and focus on the encoder/decoder split. Both families are trained with matched model parameters on the same pretraining corpus and optimization budget, and within each family, only the patch size is varied. The corpus is entirely open: it combines GIFT-Eval Pretrain with two synthetic generators, KernelSynth (Ansari et al., 2024) and our RandomPatterns pipeline, so that every model in the study can be retrained from the released code. We evaluate forecasting on the 97 tasks of GIFT-Eval, both in aggregate and at the dataset level, and then reuse the same pretrained backbones as frozen feature extractors on the 91 classification datasets of UCR-91 (Dau et al., 2019) with linear probing.

Our main findings are as follows:

- **Patch-size preference is architecture-dependent.** Smaller patches consistently improve encoder-only models, whereas decoder-only models peak at an intermediate patch size. On UCR-91 classification with frozen representations, smaller patches again help the encoder, while the decoder is hardly affected.
- **The decoder-only optimum is horizon-dependent.** At the dataset level, the patch size preferred by decoder-only models increases with the forecasting horizon, so the aggregate optimum partly reflects the benchmark's horizon distribution rather than a property of the model alone.
- **The two families occupy complementary accuracy–compute regimes.** Measured by CPU inference latency, decoder-only models provide the cheapest configurations, while encoder-only models with the finest patches are the most accurate.

We release the pretraining and evaluation framework together with the synthetic data generators as an **open codebase**[1]. Architecture, tokenization, data mixture, and training budget can be varied independently, so the framework directly supports controlled ablations beyond the one conducted here.

The remainder of the paper is organized as follows. Section 2 introduces the encoder-only and decoder-only models studied in this work and the evaluation protocol, and Section 3 reports the results: per-family fine-grained evaluations on GIFT-Eval (Aksu et al., 2024), followed by aggregated performance–efficiency comparisons between the two families.

[1]Available in supplementary material.

2 PATCH-SIZE STUDY SETUP

Isolating the effect of patch size requires removing the identified confounds: published TSFMs differ in pretraining corpus, scale, objective, and tokenization, all at once. Our setup rests on three choices. First, all models share a single pretraining recipe, with the same open corpus, optimization budget, and hyperparameters, so that within a family, only the patch size changes. Second, we use representative architectures of each paradigm and follow their published design as closely as possible. Third, we evaluate at three levels: aggregate forecasting error, dataset-level win rates split by forecasting term, which distinguishes a genuine optimum from an artifact of the benchmark's horizon distribution (Section 3.1 and Section 3.3), and linear-probing classification (Section 3.4), which tests whether the preferences extend beyond the forecasting head. An accuracy–latency analysis (Section 3.5) finally places all configurations on a common axis.

2.1 A COMMON FORECASTING INTERFACE

Probabilistic forecasting. Univariate probabilistic forecasting is the main downstream task, evaluated in Section 3, and also the pretraining task, with both models trained through the pinball loss on the predicted quantiles (Section A.3). For the encoder-only model, pretraining jointly combines forecasting and imputation: patches at the end of the context are always masked to define a forecasting horizon, while additional patches within the observed context are masked for imputation. Classification, the second downstream task, reuses the pretrained backbone as a frozen feature extractor and replaces the quantile head with linear probing. For forecasting, the input is a univariate tokenized context, and the output is a set of quantiles over a forecasting horizon. Let $\mathbf{x}_{1:C} = (x_1, x_2, \ldots, x_C) \in \mathbb{R}^C$ denote an observed context of length C and let $\mathbf{x}_{C+1:C+H}$ denote the subsequent forecasting horizon of length H. Given the context, the model predicts Q conditional quantiles for each horizon timestamp. For a set of quantile levels $\mathcal{Q} = \{\tau_1, \tau_2, \ldots, \tau_Q\} \subset (0, 1)$, the model's output is therefore $\hat{\mathbf{y}} \in \mathbb{R}^{H \times Q}$ and the goal is to have

$$\hat{\mathbf{y}}_{h,k} \approx F^{-1}_{\mathbf{x}_{C+h}|\mathbf{x}_{1:C}}(\tau_k), \quad h \in \{1, 2, \ldots H\} \text{ and } k \in \{1, 2, \ldots, Q\}, \tag{1}$$

where $F_{\mathbf{x}_{C+h}|\mathbf{x}_{1:C}}$ is the cumulative distribution function of $\mathbf{x}_{C+h}$ conditioned on $\mathbf{x}_{1:C}$.

Patching. Transformers require a token sequence, but representing each observation as a separate token is costly because of the quadratic cost of self-attention. Both model families therefore group consecutive observations into contiguous, non-overlapping patches of size P, allowing each token to capture local temporal structure while reducing the sequence length by approximately a factor of P. When C is not divisible by P, the context is zero-padded on the left, yielding $N_C := \lceil C/P \rceil$ patches.

Forecasting a horizon of length H then requires $N_H := \lceil H/P \rceil$ future patches. When H is not divisible by P, the final predicted patch is truncated to retain exactly the requested H time steps. Patch size, therefore, controls two quantities at once: the temporal resolution of each token and the number of tokens. For a fixed context, decreasing P increases the number of context tokens N_C, and for a fixed horizon, it increases the number of future patches N_H to forecast. A sweep over P at fixed C cannot separate these two effects, which is why Section 3.2 also varies the context length. Since the number of tokens drives the cost of attention, the patch size also sets the computational cost of both training and inference.

2.2 ENCODER-ONLY AND DECODER-ONLY FORECASTING

The two families represent the two dominant paradigms for patch-based transformer TSFMs. They share the patching mechanism, but differ in how information flows across tokens and how the horizon is generated: the encoder-only model uses bidirectional self-attention and predicts all future patches jointly, whereas the decoder-only model uses causal self-attention and predicts them one at a time. The encoder-only model is an implementation of PatchTST-FM (Wen et al., 2026), and the decoder-only model follows the architecture and autoregressive training procedure of Toto (Cohen et al., 2025) and TimesFM (Das et al., 2024). Keeping each family faithful to its reference design makes it representative of its paradigm, at the price of secondary differences between families. Some of these follow from the paradigm itself: a causal model cannot normalize with statistics of the whole context without leaking future values, whereas a bidirectional model can. We describe below the components that interact with patch size and defer the remaining details to Section A.

2.2.1 ENCODER-ONLY MODEL

The encoder-only family is trained as a masked reconstruction model: randomly selected patches are hidden from the input, and their values are predicted simultaneously using full self-attention. This objective matches inference, where the horizon is appended to the context as masked patches and reconstructed in one pass.

Masking and normalization. We use contiguous-patch masking (CPM), following Auer et al. (2025), to mask contiguous spans inside the context and at the end, for forecasting. Our only departure from PatchTST-FM is to mask contiguous sequences of 128 time points inside the context rather than a fixed number of patches, so that the masked span, and hence the reconstruction problem seen during training, is identical for every patch size (see Section A.1). We retain the binary mask $\mathbf{m} \in \{0,1\}^C$ where $\mathbf{m}_t = 1$ if $\mathbf{x}_t$ is neither masked nor padded and 0 otherwise.

We then normalize the input time series using mask-aware normalization, introduced by Wen et al. (2026), with mean $\mu(\mathbf{x}, \mathbf{m})$ and standard deviation $\sigma(\mathbf{x}, \mathbf{m})$ computed only using the visible time points (i.e., where $\mathbf{m}_t = 1$). The same statistics are used to transform the corresponding predictions back to the original scale. The influence of outliers is reduced by using arcsinh, as introduced by Ansari et al. (2025). The normalized series then becomes

$$\bar{\mathbf{x}} = \sinh^{-1}\left(\frac{\mathbf{x} - \mu(\mathbf{x}, \mathbf{m})}{\sigma(\mathbf{x}, \mathbf{m})}\right). \tag{2}$$

Patch embedding and backbone. For every patch, its normalized values are concatenated with the corresponding binary mask. A residual block maps the resulting vector to the encoder embedding dimension d, to which learnable positional embeddings are added. The tokenization procedure thus produces a sequence of patch embeddings $\mathbf{e} \in \mathbb{R}^{N_C \times d}$ that are further processed by the transformer layers.

Prediction. At inference time, the model appends N_H masked patches representing the forecasting horizon to the observed context patches. The hidden representation associated with each masked patch is processed by a residual output head that produces $P \times Q$ values, representing Q conditional quantiles for each of the P time points in the patch. Since the encoder processes all context and masked tokens jointly, the complete horizon is reconstructed in a single forward pass.

2.2.2 DECODER-ONLY MODEL

The decoder-only family is trained autoregressively: each patch is predicted from all preceding patches using causal self-attention, matching the patch-by-patch generation used at inference.

Normalization. Each patch is normalized using causal, prefix-dependent statistics. For the i-th patch, the mean μ_i and standard deviation σ_i are computed using only the observations up to the end of that patch. The normalized patch is therefore

$$\bar{\mathbf{p}}_i = \frac{\mathbf{p}_i - \mu_i}{\sigma_i}. \tag{3}$$

The same statistics are used to transform the corresponding predictions back to the original scale. Since patch i is used to predict patch $i+1$, the normalization statistics do not leak information from future values.

Patch embedding and backbone. For every patch, its normalized values are concatenated with the corresponding binary visibility mask. A bias-free linear projection maps the resulting vector to the decoder embedding dimension d. The tokenization procedure thus produces a sequence of patch embeddings $\mathbf{e} \in \mathbb{R}^{N_C \times d}$.

A stack of causal transformer layers processes the patch embeddings. No positional embedding is added; positional information is introduced only through RoPE (Su et al., 2024) inside the transformer layers.

Table 1: Configurations considered in our experiments.

Model family	**Patch size** P	**Context length** C
Encoder-only	4	1024
Encoder-only	8	1024, 2048
Encoder-only	16, 32, 64	1024, 2048, 4096
Decoder-only	16, 32, 64, 128, 256	2048

Prediction. During training, next-patch predictions for all positions are computed in parallel using teacher forcing. At inference time, forecasting is performed autoregressively, patch by patch. The hidden representation at each position is passed through a linear output projection producing $P \times Q$ values. The central output directly represents the median, while the remaining outputs are transformed into positive increments using $\mathrm{softplus}$. Cumulative increments are subtracted from or added to the median to obtain the lower and upper quantiles, respectively. The predicted quantiles are therefore ordered by construction.

2.3 Evaluation protocol and experimental setup

This section fixes what is common to all configurations: the pretraining recipe, the sweeps over patch size and context length, and the forecasting and classification protocols used to evaluate the pretrained models.

Pretraining recipe. All configurations are pretrained from scratch on the same mixture of 10% GIFT-Eval Pretrain (Aksu et al., 2024), 50% RandomPatterns, and 40% KernelSynth (Ansari et al., 2024). Its synthetic majority increases the diversity of temporal patterns seen during training and prevents the pretraining distribution from being dominated by the domains of the evaluation datasets. Every configuration is trained for 100k optimization steps, following Wen et al. (2026), with identical hyperparameters. The generation procedures and training details are given in Section A.4.

Model configurations. Table 1 summarizes the configurations. The two patch-size ranges overlap on $P \in \{16, 32, 64\}$ and extend in the direction each family favors, down to $P = 4$ for the encoder-only model and up to $P = 256$ for the decoder-only model. Each configuration is pretrained with multiple random seeds that were allocated sequentially: configurations that clearly underperformed in the initial runs were not replicated to the maximum number, allowing the available compute to be concentrated on competitive configurations and on estimating variability around the observed optima. All seed-level aggregates use the available runs for each configuration, whose numbers are reported in Table A.2. Full model configurations are given in Section A.

Forecasting. All the configurations are evaluated on the 97 forecasting tasks of GIFT-Eval (Aksu et al., 2024). We report normalized MASE (mean absolute scaled error) to evaluate point forecasts and normalized CRPS (continuous ranked probability score) to evaluate probabilistic forecasts. The normalization is performed with respect to the performance of the seasonal naive predictor, and aggregated results are obtained by geometrically averaging the normalized task-level scores, following the approach in Aksu et al. (2024).

Aggregate scores can be driven by a handful of datasets or by the benchmark's horizon distribution. We therefore also report the dataset win rates of each patch size, defined as the fraction of tasks on which it obtains the lowest error among the configurations in the corresponding sweep. Furthermore, we also present these win rates across GIFT-Eval's short-, medium-, and long-term forecasting categories, which contain 55, 21, and 21 tasks, respectively.

Classification. We additionally evaluate the pretrained backbones by linear probing on UCR-91, the subset of 91 univariate datasets from the UCR archive (Dau et al., 2019), using the train/test splits provided by the archive. We report classification accuracy, averaged over the 91 datasets.

Each time series goes through the model's own pipeline of patching, instance normalization, and embedding, and we extract the final layer's representation with architecture-specific pooling over

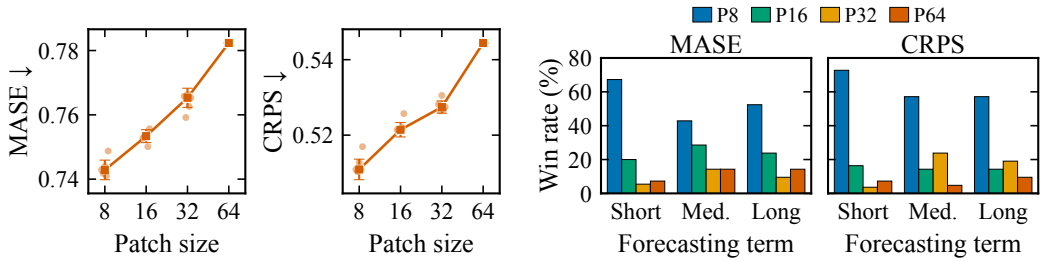

Figure 1: Effect of patch size on the encoder-only model for $C = 2048$. **Left:** normalized MASE and CRPS on GIFT-Eval. **Right:** dataset win rates by forecasting term. The encoder-only model with patch size $P = 8$ is the best in both aggregate performance and dataset win rates within each term.

valid patch tokens. For encoder-only models, we average the token representations, since bidirectional attention lets each token see the entire series. For decoder-only models, we use the final token, the only one whose receptive field covers the entire series under causal attention. A comparison of pooling strategies is given in Section B.2 (Figure B.4).

On top of these frozen representations, we fit a standardized multinomial logistic regression (`StandardScaler` followed by L-BFGS for at most 500 iterations), following the protocol of Feofanov et al. (2026). Neither the probe fitting nor the choice of layer and pooling is done on the test split. We use a linear probe as the objective is to assess the quality of the model's representation and not to maximize classification accuracy.

3 Patch size interacts with architectures across tasks

We examine how temporal patch size affects forecasting performance across the two model families. We then evaluate frozen representations on classification tasks and compare the inference cost of each model. The results reveal qualitatively different behaviors: encoder-only models consistently benefit from finer patching, whereas decoder-only models exhibit an intermediate optimum whose location depends on the forecasting horizon.

3.1 Encoder-only models benefit from finer patches

We first compare encoder-only models with patch sizes $P \in \{8, 16, 32, 64\}$ at a fixed context length $C = 2048$. Figure 1 shows a consistent deterioration in both normalized MASE and normalized CRPS as the patch size increases. Among the evaluated configurations, $P = 8$ therefore provides the most accurate point and probabilistic forecasts.

The dataset-level results show that this improvement is broadly distributed across GIFT-Eval rather than being driven by a small number of datasets. The $P = 8$ model obtains the best MASE on 57 of the 97 tasks, and the best CRPS on 64 tasks. By comparison, $P = 16$ wins on 22 and 15 tasks, respectively, while $P = 32$ wins on 8 and 11 tasks and $P = 64$ wins on 10 and 7 tasks.

The benefit of smaller patches is most pronounced for short-term forecasting, but persists across medium and long horizons. At fixed context length, smaller patches also yield longer token sequences. This suggests that finer tokenization may improve performance by preserving more local information while allowing full attention to integrate it across the entire context.

3.2 The encoder preference holds across context lengths

At a fixed context length, a smaller patch also means more tokens, so Section 3.1 does not separate the effect of patch size from that of the token count. We therefore expand the encoder-only sweep for $C \in \{1024, 2048, 4096\}$ in Figure 2.

The preference for finer patches holds at every context length. Increasing the context at a fixed patch size also helps: the gain from $C = 1024$ to $C = 2048$ is large for every patch size, whereas the gain from 2048 to 4096 is small, suggesting saturation for the present pretraining recipe. Token count

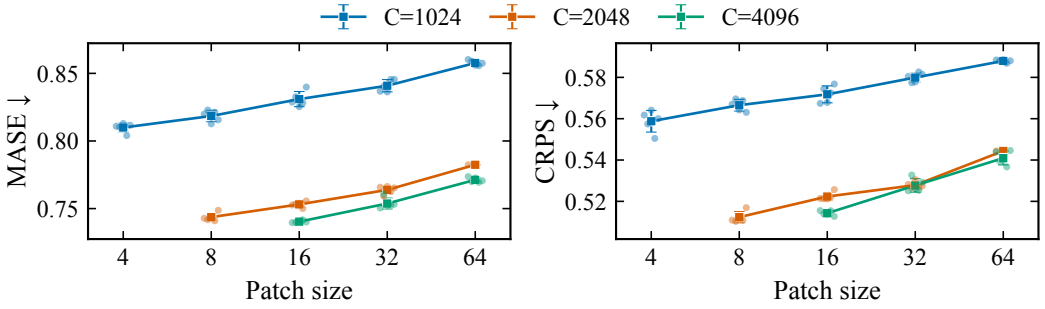

Figure 2: Encoder-only context and patch-size sweep. Within each context length, smaller patches perform better. Increasing the context length consistently improves performance up to saturation.

alone does not explain these results. At a constant 256 tokens per context, $(C, P) = (2048, 8)$ is markedly better than $(1024, 4)$ and on par with $(4096, 16)$, so the number of tokens must be read jointly with the context it covers. At fixed token count, and thus comparable per-step compute, the results suggest that beyond $C = 2048$, finer resolution and longer context can be traded off. The gains at small P can therefore not be attributed to finer temporal resolution alone, as they may be confounded with the larger token budget. Additional experiments comparing models with matched numbers of tokens processed during training are reported in Section A.6.

3.3 Decoder-only models favor an intermediate patch size

The decoder-only models exhibit a different dependence on patch size. As shown in Figure 3, their aggregate performance follows a U-shaped curve rather than improving monotonically as the patch size becomes smaller. The best overall results are obtained with $P = 64$ and both smaller and larger patches degrade performance.

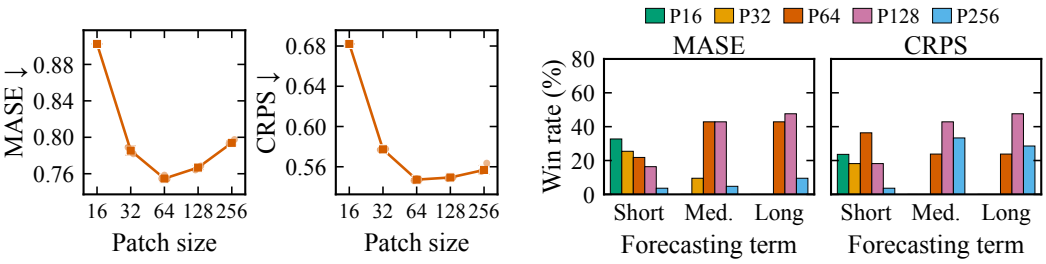

Figure 3: Effect of patch size on the decoder-only model. **Left:** aggregate MASE and CRPS on GIFT-Eval. **Right:** dataset win rates separated by forecasting term. Aggregate performance is best at $P = 64$, while the patch sizes favored by individual tasks shift as the forecasting term increases.

The preferred patch size varies with the forecasting horizon, with larger patches becoming more competitive as the horizon increases. For short-term tasks, $P = 16$ achieves the most MASE wins (18/55), while $P = 64$ achieves the most CRPS wins (20/55). In contrast, $P = 128$ performs best on medium- and long-term tasks, obtaining 9/21 and 10/21 MASE and CRPS wins, respectively.

These results explain why the intermediate patch size $P = 64$ is the best across aggregate metrics: it is a compromise between the smaller patches favored by short-term tasks and the larger patches favored at longer horizons. The aggregate optimum thus partly reflects the horizon distribution of GIFT-Eval, and a decoder-only model targeting a narrower horizon range would prefer a different patch size.

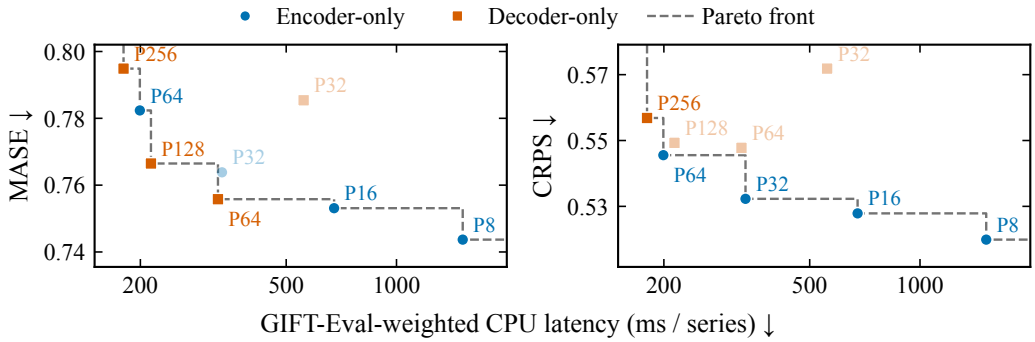

Figure 5: Normalized MASE (left) and CRPS (right) against CPU latency for encoder-only and decoder-only models for various values of the patch size P. The latency is computed from evaluations across GIFT-Eval's horizon lengths and averaged according to GIFT-Eval's proportions. Each point is the mean over pretraining seeds.

3.4 The encoder behavior transfers to classification

We next examine how temporal patch size affects classification performance across the two model families introduced in Section 2.2. The results in Figure 4 show that encoder-only models benefit substantially from smaller patches, consistent with their forecasting behavior. For the encoder, mean accuracy on UCR-91 decreases from 83.70% at a patch size of 8 to 80.44% at a patch size of 64. Decoder-only models are largely insensitive to patch size at this scale; their accuracies range from 80.61% to 80.94% across the evaluated patch sizes. Additional experiments with ∼35M-parameter models show the same trend for encoder models, and an intermediate optimum at $P = 64$ that is only visible at this smaller scale for decoder models. Full results are shown in Section B.2.

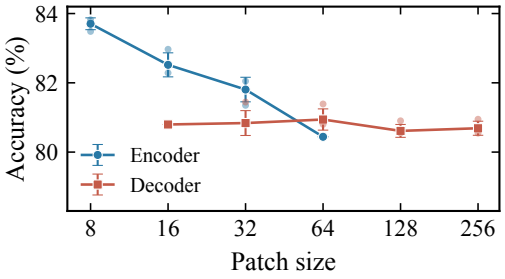

Figure 4: Effect of patch size on UCR-91 classification accuracy for the encoder-only and decoder-only models, evaluated using logistic-regression probes on frozen representations. Each point shows the median over pretraining seeds of the mean accuracy across the 91 datasets.

3.5 Accuracy–compute trade-off

Figure 5 plots the forecasting performance against the CPU latency obtained for random time series using the GIFT-Eval horizon lengths. Patch size has a large effect on the inference cost because smaller patches produce longer token sequences for the same horizon length.

The Pareto front contains configurations for which no other evaluated model is both more accurate and less computationally expensive at inference. Decoder-only models occupy the low-compute region of the front, with $P = 256$ providing the cheapest alternatives, while $P = 64$ offers the best decoder-only accuracy. Encoder-only models extend the front towards higher accuracy, with $P = 8$ achieving the lowest MASE and CRPS overall. However, moving from $P = 16$ to $P = 8$ more than doubles the training compute (Figure A.1) for a comparatively modest improvement in accuracy, indicating diminishing returns.

Since GIFT-Eval is focused on short-term forecasting, the aggregate latencies reported in Figure 5 are dominated by short forecasting horizons, for which the computational advantage of decoder-only models is limited. To assess this advantage at longer horizons, Figure 6 reports the inference latency per time series as a function of the prediction length. For both model families, horizons exceeding the output of a single forward pass are forecast autoregressively: at each step, the predicted median is appended to the context and serves as input for the subsequent step. Decoder-only models generate one patch per step, consistent with standard next-patch forecasting. Encoder-only models generate

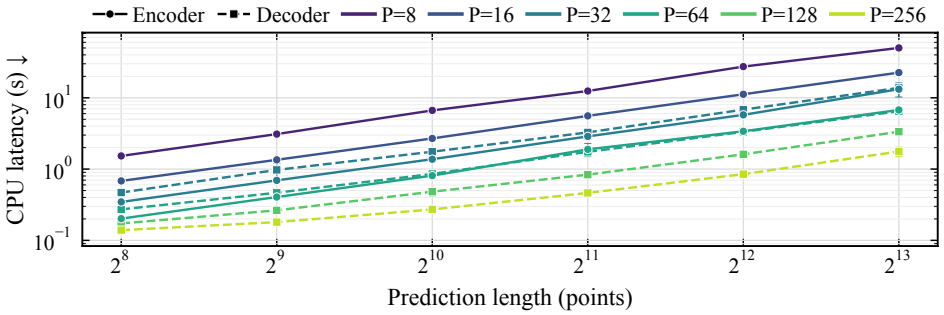

Figure 6: CPU inference latency per time series as a function of the prediction length for encoder-only and decoder-only models with different patch sizes, measured at batch size 1. Horizons exceeding a single forward pass are forecast autoregressively by feeding back the predicted median, with 256 points generated per step for encoder-only models and one patch per step for decoder-only models.

256 points per step, corresponding to the average number of masked points reconstructed at the end of the context during training.

Beyond GIFT-Eval horizons, decoder models are up to 30× faster because they tolerate larger patches; at equal patch size, both families have similar latency. This comparison concerns cost only and we do not measure accuracy. More details about the latency experiments and the same Pareto fronts against training compute can be found in Section A.5.

4 Conclusions and Limitations

We present a controlled study of how patch size affects the performance of encoder-only and decoder-only patch-based TSFMs for forecasting and classification. Despite their common use of temporal patches, the two architectures exhibit qualitatively different behavior. Across 97 GIFT-Eval tasks, finer patches consistently improve encoder-only forecasting within the evaluated range, whereas decoder-only models achieve their best aggregate performance at an intermediate patch size. Moreover, the preferred decoder patch size increases with the forecasting horizon, showing that the aggregate optimum partly reflects the horizon distribution of the evaluation benchmark. On UCR-91, frozen-feature classification also benefits from smaller encoder patches, while the decoder is comparatively insensitive to patch size.

The context-length experiments further show that patch size and context should not be considered independently. Longer contexts improve encoder forecasting, with diminishing gains beyond $C = 2048$ observations. The efficiency results reveal complementary operating regimes: Under the measured CPU inference conditions, decoder configurations provide lower-latency options, while encoder configurations achieve the highest forecasting accuracy. Neither architecture nor patch size is universally optimal, and these results support selecting patch size jointly with the model family, target horizons, and computational constraints.

Limitations. Our main findings concern the evaluated model families, patch-size ranges, and pretraining mixture, with forecasting experiments concentrated at approximately 260M parameters. The 35M-parameter runs in Section B indicate that the encoder optimum shifts from $P = 8$ to $P = 16$ at smaller scale, so the preferred patch size should be expected to depend on model size. In particular, the study covers only transformer backbones: whether recurrent or state-space TSFMs such as TiRex (Auer et al., 2025) or FlowState (Graf et al., 2026) share the patch-size preference of either family remains open.

The two families necessarily differ in training formulation, normalization, positional encoding, and prediction heads, so cross-family differences cannot be attributed to attention patterns alone. Furthermore, inference latency is measured at batch size 1 on a single laptop CPU and the resulting Pareto front may vary across hardware, numerical precision, software implementations, and batching regimes.

AI use statement. We used generative AI tools to assist with implementing methods, writing and debugging code, creating and revising scientific figures, and drafting and editing the manuscript. The authors reviewed all AI-assisted outputs and take responsibility for the final content of this work, including its text, code, figures, and scientific claims.

Reproducibility statement. The model definitions and forecasting procedures are described in Section 2, with architecture specifications and the training objective in Section A. The pretraining data mixture, synthetic data generation procedure, and optimization settings are documented in Section A.4. Section 2.3 describes the forecasting and classification evaluation protocols, including metrics, aggregation, and linear probing. The numbers of pretraining seeds per configuration are reported in Tables A.2 and B.2, and Section B documents the small-model run. Hardware and inference timing procedures are provided in Section A.5. The supplementary material includes the pretraining and evaluation framework and synthetic data generators.

REFERENCES

T. Aksu, G. Woo, J. Liu, X. Liu, C. Liu, S. Savarese, C. Xiong, and D. Sahoo. GIFT-eval: A benchmark for general time series forecasting model evaluation. In *NeurIPS Workshop on Time Series in the Age of Large Models*, 2024.

S. K. Ankireddy, N. Seleznev, N. H. Nguyen, Y. Wu, S. Kumar, F. Huang, and C. B. Bruss. Timesqueeze: Dynamic patching for efficient time series forecasting. In *Recent Advances in Time Series Foundation Models Have We Reached the 'BERT Moment'?*, 2025.

A. F. Ansari, L. Stella, A. C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor, J. Zschiegner, D. C. Maddix, H. Wang, M. W. Mahoney, K. Torkkola, A. G. Wilson, M. Bohlke-Schneider, and B. Wang. Chronos: Learning the language of time series. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. Expert Certification.

A. F. Ansari, O. Shchur, J. Küken, A. Auer, B. Han, P. Mercado, S. S. Rangapuram, H. Shen, L. Stella, X. Zhang, M. Goswami, S. Kapoor, D. C. Maddix, P. Guerron, T. Hu, J. Yin, N. Erickson, P. M. Desai, H. Wang, H. Rangwala, G. Karypis, Y. Wang, and M. Bohlke-Schneider. Chronos-2: From univariate to universal forecasting, 2025.

A. Auer, P. Podest, D. Klotz, S. Böck, G. Klambauer, and S. Hochreiter. TiRex: Zero-Shot Forecasting Across Long and Short Horizons with Enhanced In-Context Learning. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*, 2025.

B. Cohen, E. Khwaja, Y. Doubli, S. Lemaachi, C. Lettieri, C. Masson, H. Miccinilli, E. Ramé, Q. Ren, A. Rostamizadeh, J. O. du Terrail, A.-M. Toon, K. Wang, S. Xie, Z. Xu, V. Zhukova, D. Asker, A. Talwalkar, and O. Abou-Amal. This time is different: An observability perspective on time series foundation models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

A. Das, W. Kong, R. Sen, and Y. Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning*, 2024.

H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. J. Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6:1293–1305, 2019.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

K. Feng, S. Lan, Y. Fang, W. He, S. Lu, S. Gu, L. Ma, X. Lu, and K. Ren. Kairos: Toward adaptive and parameter-efficient time series foundation models, 2025.

V. Feofanov, S. Wen, S. Xie, S. Roschmann, M. Alonso, H. Guo, R. Ilbert, M. Tiomoko, Q. Bouniot, Z. Akata, L. Pan, J. Zhang, and I. Redko. Mantis: Lightweight foundation model for time series classification. In *Forty-third International Conference on Machine Learning*, 2026.

L. Graf, T. Ortner, S. Woźniak, and A. Pantazi. Flowstate: Sampling-rate-equivariant time-series forecasting. In *Forty-third International Conference on Machine Learning*, 2026.

T. Limisiewicz, A. Pagnoni, S. Iyer, M. Lewis, S. Mehta, A. Liu, M. Li, G. Ghosh, and L. Zettlemoyer. Compute optimal tokenization. 2026.

C. Liu, T. Aksu, J. Liu, X. Liu, H. Yan, Q. Pham, S. Savarese, D. Sahoo, C. Xiong, and J. Li. Moirai 2.0: When less is more for time series forecasting. *arXiv preprint arXiv:2511.11698*, 2025.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

W. Peebles and S. Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision*, 2023.

J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568(C), Feb. 2024. ISSN 0925-2312.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Thirty-first International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

F. Wang, Y. Yu, G. Wei, W. Shao, Y. Zhou, A. Yuille, and C. Xie. Scaling laws in patchification: An image is worth 50,176 tokens and more. *arXiv preprint arXiv:2502.03738*, 2025.

Y. Wen, W. M. Gifford, C. Reddy, L. M. Nguyen, J. Kalagnanam, and A. A. Julius. Revisiting the generic transformer: Deconstructing a strong baseline for time series foundation models. *arXiv preprint arXiv:2602.06909*, 2026.

G. Woo, C. Liu, A. Kumar, C. Xiong, S. Savarese, and D. Sahoo. Unified training of universal time series forecasting transformers. In *Forty-first International Conference on Machine Learning*, 2024.

A. Yu, D. Maddix, B. Han, X. Zhang, A. F. Ansari, O. Shchur, C. Faloutsos, A. G. Wilson, M. W. Mahoney, and B. Wang. Understanding transformers for time series: Rank structure, flow-of-ranks, and compressibility. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *International Conference on Learning Representations*, volume 2026, pages 42449–42489, 2026.

F. Zucchi, Y. Xie, C. Zhang, K. Luo, T. Lampert, and Z. Li. Adaptive patching is harder than it looks for time-series forecasting, 2026.

A Further experimental details and more experiments

This section provides additional implementation details for the two model families described in Section 2.2. The encoder-only model is an implementation of PatchTST-FM (Wen et al., 2026). Its only modification to the original training setup is the use of a fixed number of masked time points for each contiguous block rather than a fixed number of masks. The decoder-only model follows the architecture and autoregressive training procedure of Toto (Cohen et al., 2025) and TimesFM (Das et al., 2024).

A.1 Encoder-only PatchTST-FM

Fixed-length contiguous masking. The encoder-only model is trained using contiguous-patch masking. The context window of C time points is divided into non-overlapping, patch-aligned spans, and spans are masked uniformly at random until about 40% of the window is hidden. In addition, the last $\ell \sim \mathcal{U}(0, 4S)$ points are masked, where S is the span length, so that part of the objective is

forecasting. Our only change to PatchTST-FM is how the span length is set. PatchTST-FM fixes it in patches ($N_{\text{cpm}} = 8$), so the masked span would grow with patch size P. We instead fix it in time points, $S = 128$, i.e., $N_{\text{cpm}} = 128/P$. With this choice, the number of masked spans, their length, the length of the tail mask, and hence the reconstruction problem seen during training are the same for every patch size.

Let $\mathbf{m} \in \{0,1\}^C$ denote the visibility mask, with $\mathbf{m}_t = 1$ for visible, non-padded observations and $\mathbf{m}_t = 0$ for masked or padded positions. The time series mean and standard deviation are computed only from visible observations:

$$\mu = \frac{\sum_{t=1}^{C} m_t x_t}{\sum_{t=1}^{C} m_t}, \qquad \sigma = \sqrt{\frac{\sum_{t=1}^{C} m_t (x_t - \mu)^2}{\sum_{t=1}^{C} m_t}}. \tag{4}$$

The visible values are then normalized and transformed using $\operatorname{arcsinh}$:

$$\bar{x}_t = \operatorname{arcsinh}\left(\frac{x_t - \mu}{\sigma}\right). \tag{5}$$

Masked and padded positions are set to zero after normalization.

Embedding and transformer backbone. Each normalized patch $\bar{\mathbf{p}}_i \in \mathbb{R}^P$ is concatenated with its patch-level visibility mask $\mathbf{m}_i \in \{0,1\}^P$. A residual embedding block maps the resulting $2P$-dimensional vector to the model dimension d:

$$\mathbf{e}_i = f_{\text{enc}}\left(\begin{bmatrix}\bar{\mathbf{p}}_i \\ \mathbf{m}_i\end{bmatrix}\right) + \mathbf{r}_i, \tag{6}$$

where $\mathbf{r}_i$ is a learned positional embedding. The complete sequence is processed using full bidirectional self-attention.

Prediction. During training, the output head reconstructs the masked patches, and the loss is evaluated only at masked, non-padded time points. At inference time, $N_H = \lceil H/P \rceil$ masked patches are appended to the observed context. Full attention jointly processes the observed and masked patches, allowing the complete forecasting horizon to be predicted in a single forward pass. The final patch is truncated when H is not divisible by P.

A.2 Decoder-only model

Causal normalization. For each patch position i, the model computes normalization statistics from all observations up to the end of that patch:

$$\mu_i = \frac{1}{iP}\sum_{t=1}^{iP} x_t, \qquad \sigma_i = \sqrt{\frac{1}{iP}\sum_{t=1}^{iP}(x_t - \mu_i)^2}. \tag{7}$$

The observations in patch i are normalized as

$$\bar{x}_t = \frac{x_t - \mu_i}{\sigma_i}, \qquad (i-1)P < t \le iP. \tag{8}$$

The same statistics are used to transform the prediction of the next patch back to the original scale:

$$\hat{x}_t = \mu_i + \sigma_i \hat{\bar{x}}_t, \qquad iP < t \le (i+1)P. \tag{9}$$

Thus, each prediction uses only statistics computed from observations available before the predicted patch.

Embedding and causal backbone. The normalized values are concatenated with their binary visibility mask and projected to the model dimension using a bias-free linear layer:

$$\mathbf{e}_i = \mathbf{W}_{\text{in}}\begin{bmatrix}\bar{\mathbf{p}}_i \\ \mathbf{m}_i\end{bmatrix}, \qquad \mathbf{W}_{\text{in}} \in \mathbb{R}^{d \times 2P}. \tag{10}$$

The embeddings are processed by a causal transformer composed of RMSNorm, grouped-query self-attention, and SwiGLU feed-forward blocks. No additive positional embeddings are used. Patch positions are represented exclusively through rotary positional embeddings applied to the query and key representations inside the attention layers.

The causal attention mask ensures that the representation used to predict patch $\mathbf{p}_j$ can depend on all preceding patches $\mathbf{p}_1, \ldots, \mathbf{p}_{j-1}$, but not on $\mathbf{p}_j$ or any later patch. During training, predictions for all target positions are computed in parallel using teacher forcing.

Monotonic quantile head. For each predicted patch, a linear output projection produces $P \times Q$ unconstrained values. The central output directly parameterizes the median. The remaining outputs are transformed into positive increments using $\mathrm{softplus}$. Starting from the median, cumulative increments are subtracted to construct lower quantiles and added to construct upper quantiles. Consequently, for ordered levels $\tau_1 < \cdots < \tau_Q$, the predictions satisfy

$$\hat{y}_{t,\tau_1} \leq \hat{y}_{t,\tau_2} \leq \cdots \leq \hat{y}_{t,\tau_Q} \tag{11}$$

by construction.

At inference time, the complete observed context is first processed to predict one future patch. The predicted median is appended to the context and used when generating the following patch. Causal normalization is recomputed over the extended context, and generation continues until N_H future patches have been produced.

Table A.1: Architecture and parameter counts of the evaluated model families. Parameter counts vary slightly with patch size and context length; ranges are reported over all evaluated configurations.

Model	d_{model}	Layers	Heads	Quantiles	Parameters
Encoder-only	1024	20	16	99	255.0-267.5M
Decoder-only	1024	20	16	9	253.4–255.9M

Table A.1 summarizes the size of the two model families. The encoder-only configurations contain approximately 255-267 million parameters, while the decoder-only configurations contain approximately 253-256 million. Note that the difference in the number of quantiles does not affect the evaluations since GIFT-Eval performances are computed on 9 quantiles (0.1 - 0.9) and classification is performed using the frozen layers before the quantile head.

A.3 Quantile training objective

Both model families are trained using the pinball loss. For a target x_t, quantile level τ, and prediction $\hat{y}_{t,\tau}$, the loss is

$$\rho_\tau(x_t - \hat{y}_{t,\tau}) = \max\left\{\tau(x_t - \hat{y}_{t,\tau}), (\tau - 1)(x_t - \hat{y}_{t,\tau})\right\}. \tag{12}$$

The objective is averaged across all quantile levels and prediction positions.

A.4 Training details

The pretraining corpus consists of GIFT-Eval Pretrain, 1 million KernelSynth (Ansari et al., 2024) and synthetically generated time series using our RandomPatterns pipeline. The RandomPatterns pipeline works as follows. In order to generate a time series of length L:

1. Sample an integer period length p between 4 and 256;
2. Sample a random trend type among *linear*, *quadratic*, *periodic* and their corresponding parameters;
3. Generate a standard random walk $\mathbf{r}$ of length p;
4. Generate a trend time series $\mathbf{t}$ of length L;
5. The generated RandomPatterns time series is $\mathbf{x}_i = \mathbf{t}_i + \mathbf{r}_{i \bmod p}$.

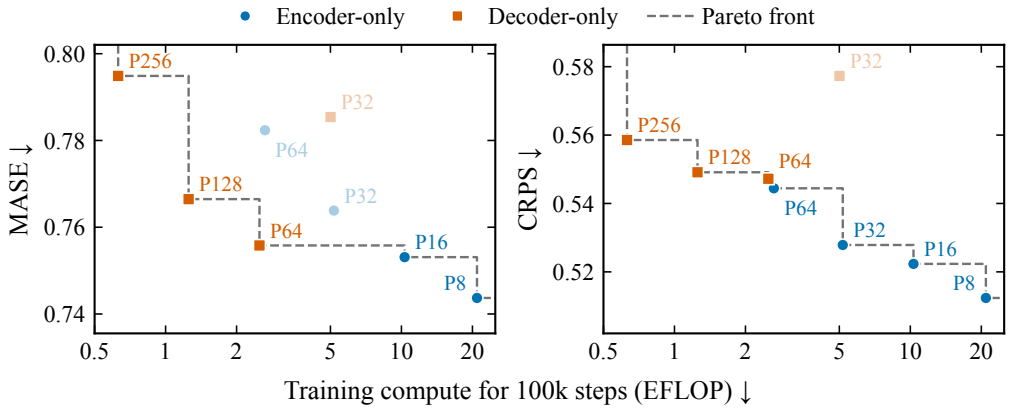

Figure A.1: Normalized MASE (left) and CRPS (right) against training compute for encoder-only and decoder-only models with various patch sizes P. Training compute denotes the number of matrix-multiplication FLOPs required to train each model for 100k steps with an effective batch size of 512 and is reported in EFLOP. Each point shows the mean performance over pretraining seeds (see Table A.2), and the dashed line indicates the Pareto front.

This pipeline provides rougher fully predictable time series with random seasonalities, preventing the model from overfitting to dataset-specific periods and also encouraging it to learn broader representations.

The dataset mixture is: 10% GIFT-Eval Pretrain, 50% RandomPatterns, and 40% KernelSynth. Each configuration is trained from scratch for 100k optimization steps using AdamW (Loshchilov and Hutter, 2019) with an effective batch size of 512. We use a peak learning rate of 3e-4 and the OneCycleLR scheduler.

Patch-size comparisons use exactly the same optimization budget, data mixture, and training hyperparameters. Consequently, changing the patch size changes the temporal tokenization and the resulting sequence length, while leaving the training recipe unchanged.

Random seeds. To account for training variability, each configuration is pretrained with several random seeds. Table A.2 lists the seeds used for every configuration.

Table A.2: Number of pretraining seeds per configuration.

Model family	**Context length**	$P = 4$	8	16	32	64	128	256
Encoder-only	1024	5	5	5	5	5	–	–
Encoder-only	2048	–	5	5	5	1	–	–
Encoder-only	4096	–	–	5	5	5	–	–
Decoder-only	2048	–	–	1	2	3	3	3

A.5 Latency and compute experiments

CPU latency in Figure 5 was measured independently on an Apple M1 Pro MacBook Pro using batch size 1, four CPU threads, one warm-up iteration, and one timed iteration. The latency benchmark reproduced the 13 distinct context–horizon conditions occurring in GIFT-Eval and combined them using their frequencies across the 97 tasks. Encoder forecasts were produced directly in one pass, whereas decoder forecasts used cached patch-recursive generation.

The latency experiments in Figure 6 were run on CPU on the same Apple M1 Pro MacBook Pro with eight cores (six performance and two efficiency cores) and 16 GB of memory. Models were randomly initialized from their evaluation configurations and executed for 5 different seeds in inference mode with batch size 1, four CPU threads, and one inter-op thread. Prediction lengths comprised 256, 512,

1024, 2048, 4096, and 8192 points. The encoder generated 256 points per autoregressive step using median feedback, whereas the decoder used its native cached patch-recursive generation procedure.

Furthermore, we provide the same Pareto fronts, against compute this time. Figure A.1 shows the training accuracy–compute trade-off. The separation between encoder-only and decoder-only dominated regions of the front is clearer in terms of FLOPs than in terms of latency: optimal models in the low-compute regime are always decoder-only, while the high-accuracy region contains encoder-only models.

A.6 Token budget experiments

At fixed context length and number of optimization steps, decreasing the patch size increases the number of tokens processed by the transformer. We define the training patch-token budget as

$$D_{\text{token}} = SB \left\lceil \frac{C}{P} \right\rceil, \tag{13}$$

where S is the number of optimization steps and B is the effective batch size (which is kept fixed at 512 in our experiments). We use the intermediate checkpoints at 50k steps to examine whether the preference for smaller patches persists when configurations are compared at matched token budgets.

Table A.3: Forecasting performance of the encoder-only models after 50k and 100k pretraining steps. MASE and CRPS are geometrically averaged over the 97 GIFT-Eval tasks. Values are reported as $\text{mean}_{\pm\text{std}}$ over pretraining seeds, and Δ denotes the mean per-seed relative change from 50k to 100k steps.

		MASE			**CRPS**		
Context	P	50k	100k	Δ	50k	100k	Δ
1024	4	$0.852_{\pm 0.013}$	$0.810_{\pm 0.003}$	-4.9%	$0.576_{\pm 0.011}$	$0.559_{\pm 0.005}$	-3.0%
1024	8	$0.882_{\pm 0.030}$	$0.818_{\pm 0.004}$	-7.1%	$0.592_{\pm 0.011}$	$0.567_{\pm 0.003}$	-4.3%
1024	16	$0.877_{\pm 0.016}$	$0.831_{\pm 0.006}$	-5.2%	$0.592_{\pm 0.004}$	$0.572_{\pm 0.004}$	-3.4%
1024	32	$0.936_{\pm 0.043}$	$0.841_{\pm 0.005}$	-10.0%	$0.620_{\pm 0.019}$	$0.580_{\pm 0.002}$	-6.3%
1024	64	$0.943_{\pm 0.026}$	$0.858_{\pm 0.002}$	-9.0%	$0.623_{\pm 0.008}$	$0.588_{\pm 0.001}$	-5.5%
2048	8	$0.778_{\pm 0.008}$	$0.744_{\pm 0.003}$	-4.4%	$0.534_{\pm 0.006}$	$0.512_{\pm 0.003}$	-4.0%
2048	16	$0.794_{\pm 0.007}$	$0.753_{\pm 0.002}$	-5.1%	$0.541_{\pm 0.010}$	$0.522_{\pm 0.002}$	-3.5%
2048	32	$0.818_{\pm 0.015}$	$0.764_{\pm 0.003}$	-6.6%	$0.562_{\pm 0.005}$	$0.528_{\pm 0.002}$	-6.1%
2048	64	0.841	0.782	-6.9%	0.568	0.544	-4.1%
4096	16	$0.777_{\pm 0.014}$	$0.740_{\pm 0.001}$	-4.7%	$0.543_{\pm 0.011}$	$0.514_{\pm 0.001}$	-5.3%
4096	32	$0.801_{\pm 0.008}$	$0.754_{\pm 0.004}$	-5.9%	$0.554_{\pm 0.003}$	$0.528_{\pm 0.003}$	-4.7%
4096	64	$0.814_{\pm 0.010}$	$0.771_{\pm 0.002}$	-5.3%	$0.565_{\pm 0.002}$	$0.541_{\pm 0.003}$	-4.2%

As shown in Table A.3, every configuration improves with additional training. From 50k to 100k steps, normalized MASE decreases by approximately 4-10% and CRPS by 3-6%. At 100k steps, smaller patches consistently perform better within each context length.

Under a matched token budget, the coarser $2P$@100k configuration outperforms the finer P@50k configuration in every comparison, reducing MASE by 3.0-8.4% and CRPS by 1.7-5.1%. This comparison favors the 100k side, since the 50k checkpoint is taken before the learning-rate schedule has annealed, so it bounds rather than measures the token-budget effect. Two observations in Table A.3 are free of this caveat, as all runs share the schedule. The fixed-step ranking is already established at 50k steps for $C \geq 2048$, and the relative improvement from 50k to 100k grows with P within each context, so coarser patches are further from convergence at equal steps. Both are consistent with the token-matched comparison of Section 3.2, where $(C, P) = (2048, 8)$ and $(4096, 16)$ tie at equal tokens and equal steps.

B Small-scale experiments

We have run experiments with smaller models in order to investigate how the main findings of our work transfer across scales. In this section, we present the main results for the architectures described

Table A.4: Token-matched comparison of encoder-only models. At fixed context length, the P@50k and $2P$@100k configurations process the same number of patch tokens during training. MASE and CRPS are geometrically averaged over the 97 GIFT-Eval tasks. Values are reported as mean$_{\pm\text{std}}$ over pretraining seeds, and Δ denotes the relative change of $2P$@100k with respect to P@50k.

			MASE			CRPS		
Context	P	$2P$	P@50k	$2P$@100k	Δ	P@50k	$2P$@100k	Δ
1024	4	8	$0.852_{\pm 0.013}$	$0.818_{\pm 0.004}$	-4.0%	$0.576_{\pm 0.011}$	$0.567_{\pm 0.003}$	-1.7%
1024	8	16	$0.882_{\pm 0.030}$	$0.831_{\pm 0.006}$	-5.8%	$0.592_{\pm 0.011}$	$0.572_{\pm 0.004}$	-3.5%
1024	16	32	$0.877_{\pm 0.016}$	$0.841_{\pm 0.005}$	-4.1%	$0.592_{\pm 0.004}$	$0.580_{\pm 0.002}$	-2.0%
1024	32	64	$0.936_{\pm 0.043}$	$0.858_{\pm 0.002}$	-8.4%	$0.620_{\pm 0.019}$	$0.588_{\pm 0.001}$	-5.1%
2048	8	16	$0.778_{\pm 0.008}$	$0.753_{\pm 0.002}$	-3.2%	$0.534_{\pm 0.006}$	$0.522_{\pm 0.002}$	-2.2%
2048	16	32	$0.794_{\pm 0.007}$	$0.764_{\pm 0.003}$	-3.8%	$0.541_{\pm 0.010}$	$0.528_{\pm 0.002}$	-2.5%
2048	32	64	$0.818_{\pm 0.015}$	0.782	-4.3%	$0.562_{\pm 0.005}$	0.544	-3.1%
4096	16	32	$0.777_{\pm 0.014}$	$0.754_{\pm 0.004}$	-3.0%	$0.543_{\pm 0.011}$	$0.528_{\pm 0.003}$	-2.8%
4096	32	64	$0.801_{\pm 0.008}$	$0.771_{\pm 0.002}$	-3.8%	$0.554_{\pm 0.003}$	$0.541_{\pm 0.003}$	-2.3%

in Table B.1. All the models have been trained for 100k steps using the same hyperparameters as before (see Section A.4), closely following PatchTST-FM masking scheme.

The number of pretraining seeds used for each small-model configuration is reported in Table B.2. As in the main experiments, configurations that clearly underperformed in the initial runs were not replicated to the maximum number of seeds. We note that one seed for the small decoder-only model with $P = 16$ did not converge and has been removed from the experiments for being an outlier.

Table B.1: Architecture and parameter counts of the small model families. Parameter counts vary slightly with patch size and context length; ranges are reported over all evaluated configurations.

Model	d_{model}	Layers	Heads	Quantiles	Parameters
Encoder-only	512	11	8	9	35.8–41.9M
Decoder-only	512	11	8	9	35.4–36.8M

B.1 Forecasting

We first examine whether the patch-size trends observed at 260M parameters persist at the smaller 35M scale. For the encoder-only models, the aggregate errors follow a U-shaped curve rather than decreasing monotonically with the patch size. As shown in Figure B.1, $P = 16$ achieves the lowest aggregate MASE and CRPS, while $P = 8$ and $P = 16$ account for most task-level wins across all forecasting terms.

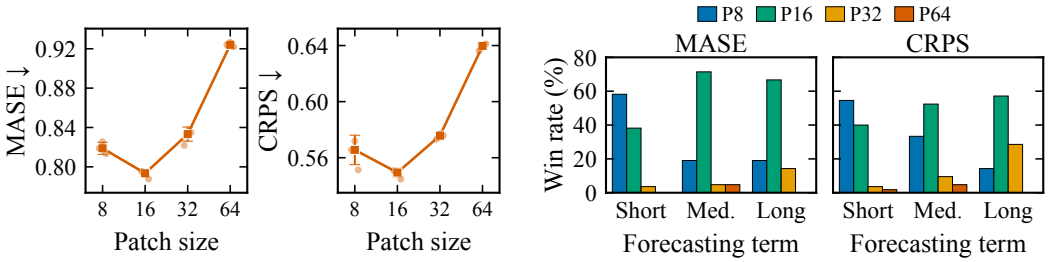

Figure B.1: Effect of patch size on the 35M-parameter encoder-only model for $C = 2048$. **Left:** normalized MASE and CRPS on GIFT-Eval. **Right:** dataset-level win rates by forecasting term. Aggregate performance is best at $P = 16$, while $P = 8$ and $P = 16$ account for most task-level wins across all forecasting terms. Results are aggregated over pretraining seeds.

Table B.2: Number of pretraining seeds per configuration for the small models.

Model family	**Context length**	$P = 8$	16	32	64	128	256
Encoder-only	2048	3	3	3	3	–	–
Decoder-only	2048	–	2	3	3	3	3

This differs from the monotonic preference for smaller patches observed in the 260M-parameter encoder. One possible explanation is a scale-dependent trade-off between temporal resolution and sequence compression. For $C = 2048$, decreasing the patch size from $P = 16$ to $P = 8$ doubles the number of input tokens from 128 to 256. Although this finer tokenization preserves more local information, it also requires the backbone to integrate information over a longer sequence. The 35M-parameter model may therefore benefit more from the additional compression provided by $P = 16$, whereas the 260M-parameter model may have sufficient capacity to exploit the finer resolution of $P = 8$. We treat this explanation as a hypothesis, since the experiments were not designed to isolate the interaction between model capacity and patch size.

The decoder-only models show a qualitatively similar dependence on patch size at both scales. As shown in Figure B.2, aggregate performance at 35M parameters is again best at the intermediate patch size $P = 64$. The dataset-level results also reproduce the horizon-dependent pattern observed at 260M parameters: smaller patches are favored more frequently for short-term forecasting, whereas larger patches become more competitive as the forecasting horizon increases.

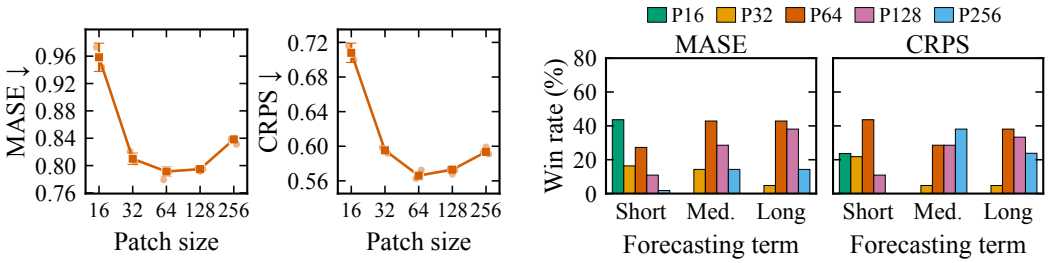

Figure B.2: Effect of patch size on the 35M-parameter decoder-only model for $C = 2048$. **Left:** normalized MASE and CRPS on GIFT-Eval. **Right:** dataset-level win rates by forecasting term. Aggregate performance is best at $P = 64$, while the patch sizes most frequently yielding the lowest task-level errors tend to increase with the forecasting horizon. Results are aggregated over pretraining seeds.

B.2 Classification

In the case of classification, the small models exhibit similar behaviors depending on the patch size as the original models. Indeed, Figure B.3 shows the results of mean classification accuracy across the 91 UCR datasets and we observe again that the encoder-only models benefit from a smaller patch size, while the inverted-U-shaped curve for decoder-only models is more pronounced than for the original models.

Table B.3 presents the full classification results for both the small models and the original ones.

Layer and pooling analysis. Figure B.4 reports the linear-probe accuracy of the small models with $P = 32$ across depth for three pooling strategies: mean, last-token, and element-wise max pooling. The encoder-only model attains its highest accuracy with max pooling at intermediate depth (82.4%), whereas the decoder-only model performs best with last-token pooling (79.9%). Both pretrained models remain well above their randomly initialized counterparts at every depth beyond the patch embedding. This analysis is not used for model selection: the results in the main text use the final layer with mean pooling for encoder-only models and last-token pooling for decoder-only models, both fixed a priori as standard strategies. The encoder-only accuracies reported in the main text may therefore underestimate the performance achievable with a tuned probe.

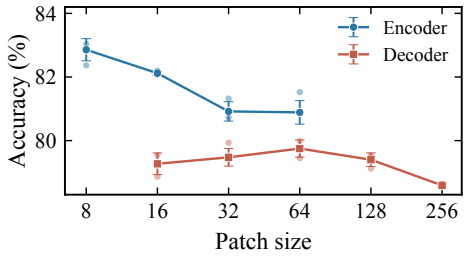

Figure B.3: Effect of patch size on UCR-91 classification accuracy for 35M-parameter encoder-only and decoder-only models, evaluated using logistic-regression probes on frozen representations. Each point shows the median over pretraining seeds of the mean accuracy across the 91 datasets. Encoder-only models perform best with the smallest patches, whereas decoder-only models exhibit an intermediate optimum at $P = 64$.

Table B.3: UCR-91 classification accuracy (%) against patch size, using frozen features and a logistic-regression probe (encoder: mean pooling; decoder: last token). Each entry is the median over pretraining seeds of the mean test accuracy across the 91 datasets, $\pm$ the standard deviation across seeds. **Bold** marks the best patch size for each model. A dash (–) means the configuration was not trained.

		Patch size					
Size	Model	8	16	32	64	128	256
Small (∼35M)	Encoder-only	$\mathbf{82.86 \pm 0.35}$	82.12 ± 0.05	80.92 ± 0.31	80.89 ± 0.37	–	–
	Decoder-only	–	79.27 ± 0.34	79.48 ± 0.28	$\mathbf{79.75 \pm 0.27}$	79.40 ± 0.21	78.59 ± 0.03
	Encoder-only (random init)	–	–	72.36	–	–	–
	Decoder-only (random init)	–	–	71.24	–	–	–
Original (∼260M)	Encoder-only	$\mathbf{83.70 \pm 0.17}$	82.52 ± 0.35	81.81 ± 0.35	80.44	–	–
	Decoder-only	–	–	80.84 ± 0.36	$\mathbf{80.94 \pm 0.31}$	80.61 ± 0.19	80.69 ± 0.20

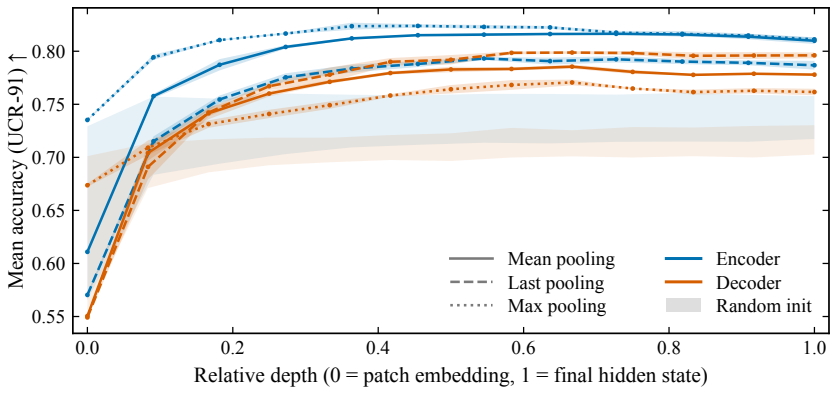

Figure B.4: Linear-probe accuracy on UCR-91 across depth for mean (solid), last-token (dashed) and max (dotted) pooling, for the small encoder-only (blue) and decoder-only (orange) models with $P = 32$ and $C = 2048$. Relative depth 0 is the patch embedding and 1 the final hidden state; the decoder includes one additional stage (its final normalization). Lines show the mean over three pretraining seeds and dark bands $\pm$ one standard deviation. Light bands show the min–max over pooling strategies of the randomly initialized models.