BYTE-PAIR PATCHING: CONTROLLABLE DYNAMIC TOKENIZATION FOR TIME-SERIES FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compression of input data, usually handled at the tokenization stage, is crucial for scaling time-series foundation models without incurring prohibitive computational costs. While static patch-based compression matches the performance of expensive point-wise tokenization, recent works have shown the scope for adaptive techniques that vary the patch size based on statistical properties of input data like entropy or frequency. To this end, inspired by Byte-Pair Encoding (BPE) for language, authors in Götz et al. (2025) expand an initial vocabulary of discretized numerical values by fusing them into *motifs* based on the frequency of occurrence in the training data. This method presents promising improvements in performance but only offers modest improvements to compression rates. In this paper, we introduce Byte-Pair Patching (BPP), a novel tokenizer that overcomes this limitation by applying BPE on fixed-size patches rather than point-wise observations, thus guaranteeing higher compression rates by design. Further, BPP also enables test-time control over compression, allowing a single pre-trained checkpoint to operate at different levels of input compression. Across GIFT-Eval, and 20 commonly used datasets, BPP provides up to $18.54\times$ increased compression versus point-wise tokenization and remains competitive relative to vanilla BPE and fixed-size patching baselines while improving inference latency by up to 25% compared to the latter.

1 INTRODUCTION

The success of foundation models in language and vision has spurred growing interest in extending this paradigm to other modalities. Time series are a natural candidate for this paradigm as pretraining on large, diverse collections of temporal data can enable models to transfer to unseen datasets with little or no task-specific adaptation. Recent time-series foundation models including Chronos and Chronos-2 (Ansari et al., 2024; 2025), TimesFM (Das et al., 2024), MOIRAI (Woo et al., 2024), Lag-Llama (Rasul et al., 2023), MOMENT (Goswami et al., 2024), and Timer (Liu et al., 2024) have demonstrated promising zero-shot forecasting performance across domains. Yet, unlike language modeling, where tokenization conventions are well established, time-series modeling still lacks a standard mechanism for converting continuous signals into tokens.

Many forecasting tasks benefit from long historical contexts, as extending the lookback window can improve predictive performance (Nie et al., 2022). Point-wise tokenization, however, makes such contexts costly. Representing every observation as a separate token preserves temporal resolution but provides no sequence compression. Since the computational and memory costs of standard self-attention scale quadratically with sequence length L (Vaswani et al., 2017; Wu et al., 2021), this representation quickly becomes impractical for the long contexts and large corpora used to train time-series foundation models.

A widely adopted alternative is fixed patching, popularized for time-series Transformers by Nie et al. (2022). Each non-overlapping block of P consecutive observations is mapped to a single token, reducing the sequence length from L to $\lceil L/P \rceil$ and substantially lowering the cost of the Transformer backbone. However, P must be fixed in advance, even though the appropriate granularity can depend on the sampling frequency, periodicity, and local complexity of the signal. This limitation also arises within individual time series. Slowly varying regions may be represented compactly, whereas rapidly changing regions may require finer resolution. A uniform patch size therefore allocates tokens

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.

inefficiently by assigning too many tokens to redundant regions while compressing information-dense regions too aggressively.

A growing body of work seeks to overcome the limitations of fixed patching through dynamic tokenization (Wang et al., 2025; Ankireddy et al., 2026; Wu et al., 2026; Abeywickrama et al., 2025). Existing approaches typically achieve adaptivity through auxiliary boundary-estimation models or multi-scale encoders, introducing additional parameters, computation, or training stages. This overhead weakens a central advantage of patching, namely its simple and scalable reduction of sequence length. However, these adaptive approaches do not consistently outperform simple fixed patching, particularly as the backbone model scales.

A promising direction introduced by Götz et al. (2025) adapts Byte-Pair Encoding (BPE), originally proposed by Sennrich et al. (2016), to time-series data, using recurring discrete patterns to construct variable-length tokens. However, it relies on point-wise quantization, which discards fine-grained signal information, and achieves substantially lower compression than commonly used fixed-patching configurations.

In this work, we introduce **Byte-Pair Patching (BPP)**, a dynamic time-series tokenizer that combines the efficiency of fixed patching with the adaptive pattern discovery of BPE. BPP divides the signal into non-overlapping patches of length P and assigns each patch a discrete ID using K-means (Lloyd, 1982). BPE then applies its vocabulary and merges the IDs into variable-length motifs. The IDs are used only to determine motif boundaries, while the final token embeddings are constructed from the continuous values within each motif. This design combines a base compression of approximately $P\times$ with additional input-dependent compression from BPE. During inference, an occurrence-count threshold determines which merge rules are applied, allowing a single checkpoint to support different compression levels and compute budgets without retraining.

Our main contributions are

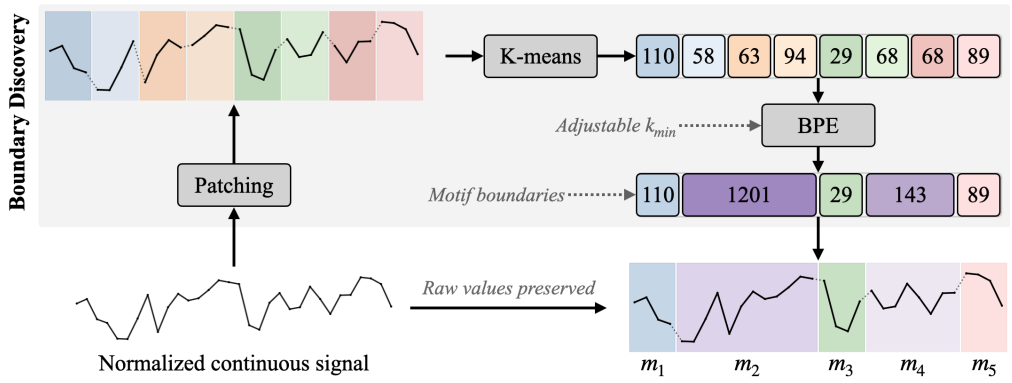
- We introduce BPP, a dynamic tokenizer that combines fixed patching with BPE-based motif discovery. Fixed patching provides a base $P\times$ compression, while BPE adds input-dependent compression.
- We preserve continuous signal information by using discrete IDs only to discover motif boundaries and constructing embeddings directly from the corresponding observations.
- We introduce a test-time compression control, allowing a single checkpoint to provide an adjustable performance–compute trade-off without retraining.
- Across GIFT-Eval and fixed-window forecasting on 20 datasets, BPP achieves competitive accuracy with substantial compression, reaching $18.5\times$ average compression on GIFT-Eval and up to 25% faster inference than fixed patching with patch size 16.

2 Related Work

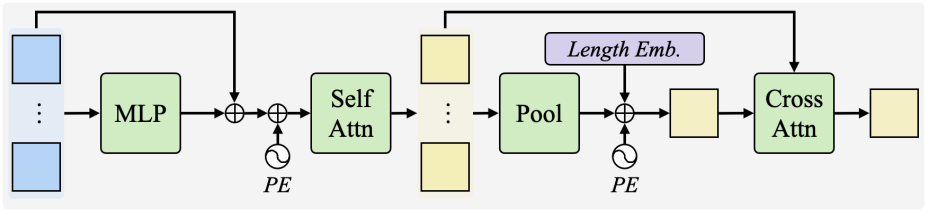
Early time-series foundation models (TSFMs) adapted simple tokenization strategies. Chronos (Ansari et al., 2024) uses sample-level tokenization, scaling, and quantizing each observation into a discrete token. Although this design allows standard language-model architectures to process time series directly, it produces one token per observation and becomes costly for long contexts. Fixed patching, popularized by Nie et al. (2022), alleviates this problem by grouping every P consecutive observations into a single token embedding. This approach has been widely adopted by recent TSFMs, including Chronos-2 (Ansari et al., 2025), because it substantially reduces sequence length and computation. However, the patch size remains fixed after training and cannot adapt to variations across or within signals.

Recently, a flurry of work has emerged to address the rigidity of fixed patching through adaptive tokenization. LightGTS (Wang et al., 2025) sets the patch length according to the signal's intrinsic period, obtained from prior knowledge or estimated using the Fourier transform. Its flexible projection layer allows a single model to embed patches with different lengths. Although this approach supports different patch sizes across signals, it cannot adjust compression in response to local signal dynamics.

TimeSqueeze (Ankireddy et al., 2026) processes the full-resolution signal with Mamba and identifies boundaries when the change between consecutive observations exceeds a threshold determined by



(a) Discovery stage



(b) Representation stage

Figure 1: BPP overview. **(a) Discovery stage.** The normalized signal is divided into fixed-size patches, each of which is assigned discrete IDs using K-means. BPE applies its learned merge rules, with $k_{\min}$ controlling which rules are eligible. The resulting motif boundaries are applied to the corresponding patches. **(b) Representation stage.** Each motif is converted into a fixed-dimensional token through patch projection, self-attention, pooling, and cross-attention.

local signal power. It then retains the Mamba embeddings at the detected boundaries as token embeddings. Although this method can adapt to within-signal changes, its hand-designed deviation rule considers only local variations and may not capture longer-range temporal dependencies when selecting boundaries.

EntroPE (Abeywickrama et al., 2025) takes an information-theoretic approach to dynamic patching. It pretrains a lightweight causal Transformer to estimate conditional entropy at each time point and places boundaries where the entropy or its change exceeds predefined thresholds. Its adaptive encoder maps the resulting variable-length patches to fixed-dimensional representations using within-patch self-attention, max pooling, and cross-attention. Although this approach adapts boundaries to changes in predictive uncertainty, it requires a separately pretrained entropy model and an additional full-resolution pass to identify boundaries at inference.

Götz et al. (2025) adapts BPE, a widely used language-tokenization technique, to time series. Their method quantizes observations into discrete symbols and learns a vocabulary of frequent adjacent-symbol merges that compresses the signal into token IDs. Finer quantization preserves signal details but makes repeated symbol pairs less frequent, limiting BPE compression and requiring a larger vocabulary. This method therefore uses relatively coarse quantization, with 37 bins in its medium-compression configuration compared to $4,096$ in Chronos. Although the authors introduce conditional decoding to reduce quantization error when mapping predicted tokens back to continuous values, the input signal remains coarsely quantized and loses fine-grained information. Despite this loss of signal precision, its reported configurations provide less than $5\times$ average compression, compared with the $8\times$–$16\times$ compression commonly provided by fixed patching.

3 METHOD

Existing adaptive tokenizers aim to balance the trade-off between forecasting performance and compression rate, while fixed-size patching provides reliable compression efficiency but are rigid in

their design. In contrast, BPP addresses the deficiencies of both of these by combining fixed-size patching with BPE: the initial patching sets a minimum baseline compression rate, and BPE merges recurring patch sequences into variable-length motifs providing further data-dependent reduction in input length.

The tokenization of BPP works in two stages: First, the *discovery stage* patches the signal, assigns each patch a discrete identifier, and applies BPE to group frequently occurring neighboring patches into variable-length motifs. Then, the *representation stage* reverts back to the continuous input space and constructs fixed-dimensional token embeddings from the continuous observations within each motif. We detail these stages in Sections 3.1 and 3.2, and further introduce a test-time compression control in Section 3.4. Figure 1 provides an overview of our proposed pipeline. Although the formulation can be extended to multivariate patches, in this work, we evaluate only univariate time series, and treat its application to multivariate time series as a crucial future investigation.

3.1 Discovery Stage

Visualized in Figure 1a, the discovery stage divides the input time series into uniformly sized base patches, assigns each patch a discrete ID, and applies BPE to group recurring ID sequences into motifs. The resulting motif boundaries are then passed to the representation stage. We detail each operation below.

3.1.1 Normalization and Patching

We begin by normalizing and patching the input time series. Let $x = (x_1, \ldots, x_L) \in \mathbb{R}^L$ denote a series of L observations. Following Liu et al. (2022), we apply z-score normalization to make tokenization invariant to shifts in level and scale. Specifically, we set $\tilde{x}_t = \frac{x_t - \mu}{\sigma + \epsilon}$, where μ and σ are computed over observed values and ϵ is a small constant that prevents division by zero. During training, these statistics are computed over the entire available series, whereas at inference, they are computed only over the observed context. Assuming L is divisible by P, we divide the normalized signal into $N = L/P$ non-overlapping patches $p^{(1)}, \ldots, p^{(N)} \in \mathbb{R}^P$. Each patch i also carries a binary observation mask $o^{(i)} \in \{0, 1\}^P$, that indicates whether or not the corresponding value in the patch is missing.

3.1.2 Discrete Patch Identifiers

Next, we discretize the normalized patches by assigning similar patches the same identifier. Formally, we learn a codebook $\mathcal{C} = \{m_k\}_{k=1}^{C} \subset \mathbb{R}^P$ of size C by applying K-means clustering (Lloyd, 1982) to the normalized patches in the training corpus. At tokenization time, each patch $p^{(i)}$ is assigned to its nearest center:

$$e_i = \arg \min_{k \in \{1,\ldots,C\}} \left\| o^{(i)} \odot \left(p^{(i)} - m_k\right) \right\|^2,$$

where the observation mask $o^{(i)}$ ensures that only observed positions contribute to the distance. By mapping the initial set of patches $(p^{(i)})_{i=1}^{N}$ to their corresponding centroids (or alternatively, ID's) $(e_i)_{i=1}^{N}$ with each $e_i \in \{1, \ldots, C\}$, we obtain an initial discrete representation.

3.1.3 BPE Motif Merging

With a discrete ID assigned to each patch, we can now expand this initial vocabulary by merging frequently occurring sequences of IDs into variable-length motifs. To this end, we apply the Byte-Pair Encoding (BPE) algorithm, a widely used language-tokenization scheme that was recently adapted to time-series data by Götz et al. (2025). In contrast to their point-wise vocabulary, we begin with our patch-based vocabulary $\mathcal{V}_0 = \{1, \ldots, C\}$. During *tokenizer training*, BPE iteratively expands $\mathcal{V}$ by merging the most frequent adjacent pair of vocabulary elements and assigning them a new ID. In order to enable our test-time compression adaptation, we also record the number of occurrences of each pair in the vocabulary. We control the vocabulary size by limiting the number of BPE iterations using a maximum vocabulary size V. We also limit the participation of partially populated patches by leaving out any initial patches with $\|o^{(i)}\|_1 < n_{\min}^{\text{obs}}$. We use $n_{\min}^{\text{obs}} = 2$ in all experiments. Finally, we set a maximal length $L_{\max} * P$ for any element in the expanded vocabulary.

Table 1: Performance comparison on GIFT-Eval: We present the relative Mean Squared Error and Mean Absolute Error of our BPP methods against baselines (described in Section 4.1.1). Relative MSE and MAE are geometric means of performance relative to a naive seasonal baseline (lower is better) as in (Aksu et al., 2024). Bold and underline indicate the best and second-best forecasting results, respectively, per scale. Both BPP configurations share the same checkpoint.

Tokenizer	Avg. Compression (×)	Tiny			Small			Base		
		Params. (M)	Rel. MSE	Rel. MAE	Params. (M)	Rel. MSE	Rel. MAE	Params. (M)	Rel. MSE	Rel. MAE
PS	1.00	4.21	0.8173	1.0231	21.01	0.7500	0.9787	88.15	0.7439	0.9656
P4	4.00	3.24	0.7129	0.9292	19.19	0.6813	0.9005	85.61	0.6801	0.8916
P8	8.00	3.24	0.7037	0.9209	19.19	<u>0.6757</u>	**0.8888**	85.62	<u>0.6731</u>	**0.8892**
P16	16.00	3.24	**0.6908**	**0.9066**	19.21	0.6801	0.8987	85.65	0.6909	0.8949
LightGTS-16	15.83	3.37	0.7519	0.9533	19.71	0.7231	0.9332	86.79	0.7147	0.9262
LightGTS	70.95	3.39	0.7825	0.9747	19.75	0.7556	0.9533	86.85	0.7789	0.9675
TimeSqueeze	4.65	3.65	0.6959	0.9129	19.58	0.6998	0.9139	-	-	-
BPE	4.08	3.59	0.8231	1.0173	19.70	0.7521	0.9595	86.19	0.7223	0.9370
BPP-n-q	15.10	3.68	-	-	20.25	-	-	87.69	0.7256	0.9501
BPP-e-q	18.54	3.68	-	-	20.25	-	-	87.69	0.7514	0.9749
BPP-n	15.10	3.68	<u>0.6953</u>	<u>0.9116</u>	20.25	**0.6725**	<u>0.8965</u>	87.69	**0.6722**	<u>0.8899</u>
BPP-e	18.54	3.68	0.7096	0.9296	20.25	0.6959	0.9195	87.69	0.6839	0.9082

3.2 Representation Stage

We now move on to building the input token representations for each element in our motif vocabulary collected from the discovery stage. Representing these motifs using only their IDs would discard continuous variations among motif instances and introduce quantization error. To overcome the possibility of propagating this error through the forecasting model, we only use the motifs to determine token boundaries and construct each token directly from the continuous values within motif boundaries. Our results in Table 1 emphasize the benefit of this design choice (BPP outperforms its discrete counterpart BPP-q). In the remainder of this section, we detail how BPP transforms the variable-length motifs into fixed size token embeddings that can be consumed by a Transformer-based forecasting model. Our design is based on the architecture proposed by Abeywickrama et al. (2025) and performs intra-motif self-attention, pooling and cross-attention chronologically. By enforcing the maximal length of a motif to be $L_{max} * P$, we only suffer a computational overhead that is linear in the overall input sequence length L and quadratic only in L_{max}.

3.2.1 Patch Encoding & Self-Attention

To construct a fixed-dimensional token from each variable-length motif, we first embed its participating patches and apply self-attention across them. This produces a contextualized representation for each patch that incorporates information from the entire motif. Formally, in the patched raw input time series $(p^{(1)}, ..., p^N)$, suppose that the motif $\mathcal{M}_j$ spans the base patches from $p^{(a)}$ through $p^{(b)}$. We first concatenate each patch $p^{(i)} \in \{p^{(k)}\}_{k=a}^{b}$ with its respective observation mask $o^{(i)}$ and project it into a d-dimensional embedding using a residual multilayer perceptron (MLP). We then add a *global* sinusoidal positional encoding (Vaswani et al., 2017) that reflects the patch's absolute position in the full sequence and apply self-attention only across patches within this motif:

$$\forall i \in a, \ldots, b, \quad \tilde{h}_i = \mathrm{MLP}\Big([p^{(i)}; o^{(i)}]\Big) + \mathrm{PE}(i), \quad \{h_i\}_{i=a}^{b} = \mathrm{SelfAttn}\Big(\{\tilde{h}_i\}_{i=a}^{b}\Big), h_i \in \mathbb{R}^d.$$

3.2.2 Pooling

Given the contextualized patch representations produced in the previous layer, we now summarize motif $\mathcal{M}_j$ into a single vector. Unlike Abeywickrama et al. (2025), which uses max pooling, we select the representation of the motif's final patch. Since self-attention contextualizes each patch using the entire motif, this representation captures information from all constituent patches while remaining aligned with the motif endpoint. We then incorporate the motif length and global endpoint position using a learned length embedding (LE) and sinusoidal positional encoding:

$$u_j = h_b, \qquad q_j = u_j + \mathrm{LE}(b - a + 1) + \mathrm{PE}(b).$$

3.2.3 CROSS-ATTENTION

Finally, we treat q_j as a query over the contextualized patch representations $\{h_i\}_{i=a}^{b}$, which serve as both keys and values. This cross-attention step combines information from all patches in the motif to produce its final fixed-dimensional token $z_j \in \mathbb{R}^d$:

$$z_j = \mathrm{CrossAttn}\Big(\text{Query} = q_j,\ \text{Keys} = \{h_i\}_{i=a}^{b},\ \text{Values} = \{h_i\}_{i=a}^{b}\Big).$$

3.3 FORECASTING MODEL

We pass the sequence of motif tokens generated in the representation stage to a T5 encoder (Raffel et al., 2020), which produces a contextualized representation for each motif. We use the representation of the final valid motif to summarize the context and pass it to an MLP regression head that directly predicts the next H continuous observations. We train the entire model end to end using mean squared error and set $H = 64$ in all experiments.

3.4 TEST-TIME COMPRESSION CONTROL

Each learned merge rule in the BPE vocabulary has an associated occurrence count k, providing an occurrence-based mechanism for controlling compression. However, changing the eligible merge rules after training produces motif segmentations that the forecasting model may not have encountered during training. In standard BPE models, including the method of Götz et al. (2025), each motif ID is associated with a learned lookup embedding, so activating previously unused rules can introduce tokens without adequately trained representations. In contrast, BPP uses BPE only to determine the motif boundaries, while the representation stage constructs each token directly from the continuous values. Therefore, the tokenizer can represent any resulting motif.

Based on this property, we control compression at inference using a threshold $k_{\min}$. We apply only merge rules satisfying $k \geq k_{\min}$. Increasing $k_{\min}$ retains fewer rules, producing shorter motifs and more tokens, while lowering it admits additional rules and increases compression. Changing this threshold requires no new parameters or retraining, allowing a single checkpoint to support different compression–performance trade-offs. We train BPP with $k_{\min} = 20{,}000$ and evaluate the same checkpoint as **BPP-n** using $k_{\min} = 10{,}000$ and as **BPP-e** using $k_{\min} = 1{,}000$. Beyond these two operating points, we sweep $k_{\min}$ over a broader range to characterize the resulting compression–performance trade-off in Section 4.6.

4 EXPERIMENTS

We evaluate whether BPP can reduce sequence length while preserving forecasting accuracy and lowering inference cost. We use GIFT-Eval as our primary benchmark, compare against different fixed and adaptive tokenization baselines, and ablate the main design choices in our representation stage. We report additional fixed-window results on 20 commonly used datasets in Section B.2.

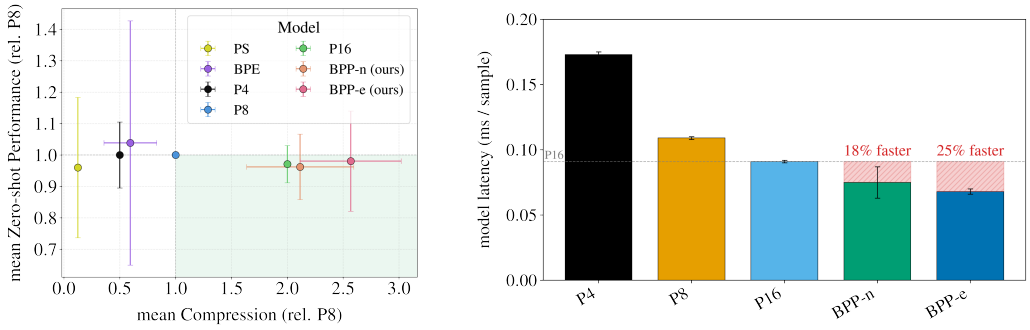
4.1 SETUP

4.1.1 BASELINES

Per-Sample (PS)– represents each observation as a separate discrete token using the mean-scale uniform-bin tokenizer from Chronos (Ansari et al., 2024).

Fixed Patching (P4/P8/P16)– divide the input into non-overlapping fixed-size patches containing 4, 8, or 16 observations (Nie et al., 2022).

LightGTS– selects a patch size for each input based on its dominant period, estimated using the Fourier transform, and maps patches of different lengths to a shared embedding space using a flexible projection layer (Wang et al., 2025). In our results, LightGTS follows the original configuration and permits patch lengths up to 96. LightGTS-16 caps the patch length at 16, providing a comparison within the compression range of the other methods.



(a) Accuracy–compression trade-off for our fixed-window forecasting setting. Metrics are relative to P8. The shaded region denotes the region with lower error and higher compression than P8.

(b) Base-model inference latency on SaugeenDay/D using a batch of 64 context windows of length 512. Bars report the mean per-sample latency over 20 timed runs following five warm-up runs, with tokenization excluded.

Figure 2: Accuracy–compression trade-off and model-side inference efficiency of BPP.

TimeSqueeze– encodes the full-resolution signal using Mamba and places boundaries where the change between consecutive observations exceeds a threshold determined by local signal power. The Mamba representations at these boundaries form the resulting tokens (Ankireddy et al., 2026).

BPE– quantizes individual observations into 37 bins and applies BPE directly to the resulting discrete sequence (Götz et al., 2025).

BPP-n/BPP-e (ours)– are two compression configurations of BPP derived from the same checkpoint, which is trained with $k_{\min} = 20{,}000$ and a K-means codebook of 64 centers. At inference, BPP-n uses $k_{\min} = 10{,}000$, while BPP-e uses $k_{\min} = 1{,}000$ for greater compression.

BPP-n-q/BPP-e-q (ours, quantized)– use the same motif boundaries and compression settings as BPP-n and BPP-e, respectively, but replace each continuous patch with its assigned K-means center before the representation stage. These variants isolate the benefit of constructing tokens from continuous rather than quantized values.

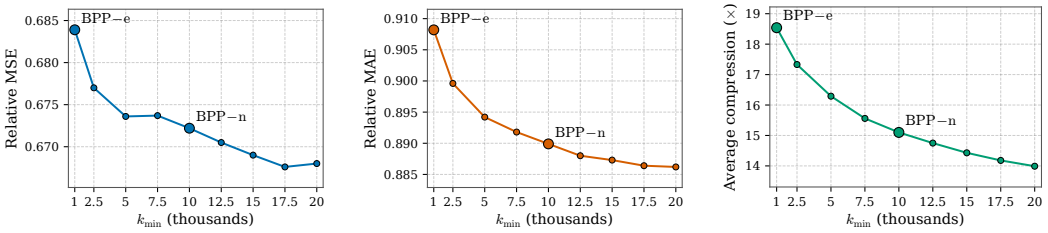
Additional implementation details for all baselines are provided in Section B.1.1.

4.1.2 Models and Training Protocol.

We use encoder-only T5 backbones at three scales, namely tiny, small, and base. At each scale, all methods use the same training data, backbone configuration, optimizer, and training steps, with only the tokenizer and its corresponding input layer varying. Following Götz et al. (2025), we construct the vocabularies for BPE and BPP from a random sample of 100,000 univariate series from the Chronos pretraining corpus (Ansari et al., 2024). All forecasting models are then trained on the full Chronos corpus for 200,000 steps. Methods without a prescribed normalization scheme use z-score normalization. Normalization statistics are computed over the complete available series during training and only over the provided context at inference. Each model predicts the next 64 observations using an MLP regression head and is trained with MSE. We otherwise follow the Chronos training configuration, with minor stability adjustments. Because the input layers differ, total parameter counts vary slightly across tokenizers and are reported in the results. Complete model and training configurations are provided in Section B.1.2.

4.2 GIFT-Eval Results

We evaluate all checkpoints on the full GIFT-Eval benchmark (Aksu et al., 2024). For horizons longer than the model's 64-step output, we forecast recursively in blocks of 64, append each prediction to the context, and retain the most recent 512 observations. Following GIFT-Eval, we report the geometric means of MSE and MAE relative to the seasonal-naive baseline across all benchmark configurations. Compression is computed globally as the total number of input observations divided by the number of resulting tokens.



(a) Relative MSE (b) Relative MAE (c) Average compression

Figure 3: Effect of $k_{\min}$ on GIFT-Eval performance using the base-scale checkpoint trained with $k_{\min} = 20{,}000$. The highlighted points correspond to BPP-n and BPP-e.

Table 1 presents the results of this experiment. As expected, PS provides no compression, while pointwise BPE achieves only $4.08\times$ compression and underperforms P4 despite their similar compression rates. LightGTS reaches $70.95\times$ compression but incurs a substantial loss in forecasting accuracy. Restricting its maximum patch length in LightGTS-16 improves its accuracy, but even then it remains consistently behind fixed patching and BPP at a comparable compression level. TimeSqueeze is competitive at the tiny scale but provides only $4.65\times$ compression and falls behind fixed patching at the small scale. We omit its base-scale results because training was both computationally costly and unstable.

BPP-n achieves the lowest relative MSE at the small and base scales. At the tiny scale, it remains close to P16, the strongest fixed-patching configuration, while providing $15.10\times$ compression. Using the same checkpoint, BPP-e increases compression to $18.54\times$, exceeding P16, with a modest reduction in forecasting accuracy. These results show that changing $k_{\min}$ at inference provides a practical compression–performance trade-off without retraining.

Finally, the quantized variants substantially underperform their continuous counterparts at the base scale. Using quantized patches increases relative MSE from 0.6722 to 0.7256 for BPP-n and from 0.6839 to 0.7514 for BPP-e. This confirms that using discrete IDs only for boundary discovery, while constructing tokens from continuous observations, preserves information important for forecasting.

4.3 Fixed-Window Evaluation and Robustness

We additionally evaluate the trained checkpoints on fixed-window forecasting across 20 commonly used datasets. Each model receives 512 context observations and predicts the next 64. Dataset descriptions, complete results across model scales, and compression ratios are provided in Section B.2.2.

To summarize the accuracy–compression trade-off, we normalize each model's forecasting error and compression relative to P8 and average the resulting ratios across datasets. Figure 2a presents the base-scale results. Both BPP configurations lie on the Pareto frontier. BPP-n achieves slightly lower average error than P8 with approximately $2.1\times$ its compression, while BPP-e increases this ratio to approximately $2.6\times$ while maintaining comparable accuracy. Additional three-seed experiments show that these conclusions are stable across initializations, as reported in Section B.2.3.

4.4 Inference Efficiency

Compression ratios do not directly determine batched inference efficiency because variable-length sequences must be padded to the longest sequence in each batch. We therefore measure model-side latency for the base-scale models using a batch of 64 SaugeenDay/D context windows, each containing 512 observations. Inputs are tokenized and padded before timing. The timed computation includes the tokenizer-specific input layer, T5 encoder, and regression head that predicts the next 64 observations.

We compile each model using `torch.compile` and run the experiment on an NVIDIA RTX 6000 GPU. After five warm-up runs, we average latency over 20 forward passes and divide the batch latency by 64 to obtain per-sample latency. We exclude tokenization to isolate the computational savings within the neural model from implementation-specific preprocessing overhead.

As shown in Figure 2b, BPP-n and BPP-e reduce latency by 18% and 25%, respectively, relative to P16. These results demonstrate that the additional compression provided by BPP translates into practical model-side speedups even after accounting for batch padding.

4.5 Representation Stage Ablations

We ablate the representation stage using three variants trained from scratch with the same tokenizer and base-scale backbone. *Max Pooling* replaces final-patch pooling with max pooling, *No Self-Attention* removes the self-attention, and *Padded MLP* replaces the representation stage with an MLP over padded motif values, masks, and lengths. Section B.2.5 provides the details of this experiment.

Table 2 reports the results. Based on these results, max pooling consistently reduces forecasting accuracy, with the largest degradation under high compression. The padded MLP performs better but remains worse than the proposed representation stage. It is also less scalable because its input dimension grows with $L_{\max}$, wastes computation on padded positions, and requires a finite maximum motif length.

Table 2: Representation-stage ablations on GIFT-Eval using base-scale models. Each trained checkpoint is evaluated under both BPP-n (Normal) and BPP-e (High) configurations. We report relative MSE and MAE (↓).

	Normal Compression		High Compression	
Representation	Rel. MSE	Rel. MAE	Rel. MSE	Rel. MAE
BPP (ours)	0.6722	0.8899	0.6839	0.9082
Max Pooling	0.6884	0.9094	0.7291	0.939
No Self-Attention	0.6651	0.8890	0.6929	0.9145
Padded MLP	0.6767	0.8926	0.7049	0.9182

Removing self-attention slightly improves performance under normal compression, indicating that within-motif self-attention is not essential in this setting. However, this variant degrades more noticeably when evaluated under high compression, where motifs are generally longer. The complete representation stage achieves the best high-compression performance and the smallest degradation between compression settings, demonstrating its effectiveness and robustness to test-time changes in motif length and segmentation.

4.6 Test-Time Compression Trade-Off

To evaluate the test-time compression control introduced in Section 3.4, we vary $k_{\min}$ from $1{,}000$ to $20{,}000$ while keeping the base-scale checkpoint fixed. This checkpoint was trained only with $k_{\min} = 20{,}000$, so lower thresholds produce segmentations that were not used during training. For each threshold, we evaluate the checkpoint on GIFT-Eval and report relative MSE, relative MAE, and average compression.

Figure 3 shows a consistent compression–performance trade-off. Lowering $k_{\min}$ progressively increases compression by admitting less frequent merge rules, while forecasting error increases gradually. In particular, reducing the threshold from $20{,}000$ to $1{,}000$ increases average compression from $13.99\times$ to $18.54\times$, while relative MSE increases from 0.6680 to 0.6839 and relative MAE from 0.8862 to 0.9082. The intermediate thresholds provide additional configurations between these endpoints. These results demonstrate that a single checkpoint can support a range of compression and performance levels at inference without retraining.

5 Discussion and Future Work

We introduced BPP, a dynamic time-series tokenizer that combines fixed-size patching with adaptive BPE boundary discovery. Our results suggest that adaptive tokenization can provide greater compression than fixed patching with competitive forecasting accuracy. We also show that separating boundary discovery from continuous representation enables test-time adjustable compression. Despite these promising results, our evaluation is limited to univariate forecasting, leaving the effectiveness of BPP for multivariate signals to future work. Moreover, because the K-means codebook and BPE vocabulary are learned from the Chronos corpus, their robustness under substantial domain shifts also remains to be established. We note that our latency analysis measures model-side computation at a single context length and excludes tokenization because the current BPE implementation is not optimized. Future work should evaluate longer contexts, develop an efficient end-to-end implementation, and adapt $k_{\min}$ automatically to individual inputs or compute budgets.

AI USE STATEMENT

We used large language models to assist with pipeline implementation, provide feedback on research methodology and experimental design, and draft and edit portions of the paper. The research ideas, experimental decisions, and final interpretations were developed and approved by the authors. AI-assisted code was integrated with established public implementations, including the Chronos codebase, and validated through unit testing, experimental evaluation, and manual inspection. All AI-generated text was reviewed and revised by the authors, and suggestions were critically assessed and discarded when deemed inappropriate or unsupported. We did not use generative AI to generate datasets, formulate or prove mathematical claims, or identify related work. The authors take full responsibility for the paper's content, claims, and artifacts.

REPRODUCIBILITY STATEMENT

To support reproducibility, the appendix provides the model and tokenizer configurations, baseline implementation details, training hyperparameters, hardware setup, evaluation procedures, and dataset-specific preprocessing used in our experiments. All datasets and preprocessing pipelines are publicly available, with their sources identified in the appendix. We also report complete dataset-level results and additional evaluations across random seeds. Although we are unable to release the source code, trained checkpoints, and tokenizer artifacts at this time, we have included the implementation and experimental details necessary to independently reproduce the reported results.

REFERENCES

Sachith Abeywickrama, Emadeldeen Eldele, Min Wu, Xiaoli Li, and Chau Yuen. Entrope: Entropy-guided dynamic patch encoder for time series forecasting. *arXiv preprint arXiv:2509.26157*, 2025.

Taha Aksu, Gerald Woo, Juncheng Liu, Xu Liu, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. Gift-eval: A benchmark for general time series forecasting model evaluation. *arXiv preprint arXiv:2410.10393*, 2024.

Sravan Kumar Ankireddy, Nikita Seleznev, Nam H Nguyen, Yulun Wu, Senthil Kumar, Furong Huang, and C Bayan Bruss. Timesqueeze: Dynamic patching for efficient time series forecasting. *arXiv preprint arXiv:2603.11352*, 2026.

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.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=jn2iTJas6h.

Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

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.

Leon Götz, Marcel Kollovieh, Stephan Günnemann, and Leo Schwinn. Byte pair encoding for efficient time series forecasting. *arXiv preprint arXiv:2505.14411*, 2025.

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

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

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*, 2024.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982. doi: 10.1109/TIT.1982.1056489.

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.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, et al. Lag-llama: Towards foundation models for probabilistic time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 1715–1725, 2016.

Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 914–921, 2020.

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

Yihang Wang, Yuying Qiu, Peng Chen, Yang Shu, Zhongwen Rao, Lujia Pan, Bin Yang, and Chenjuan Guo. Lightgts: A lightweight general time series forecasting model. *arXiv preprint arXiv:2506.06005*, 2025.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Forty-first International Conference on Machine Learning*, 2024.

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.

Yulun Wu, Sravan Kumar Ankireddy, Samuel Sharpe, Nikita Seleznev, Dehao Yuan, Hyeji Kim, and Nam H Nguyen. Dynamic tokenization via reinforcement patching: End-to-end training and zero-shot transfer. *arXiv preprint arXiv:2603.26097*, 2026.

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.

A METHOD

A.1 DISCOVERY STAGE

As illustrated in Figure 1, the discovery stage segments the time series into variable-length motifs by patching the signal, assigning discrete patch identifiers, and applying BPE.

A.1.1 NORMALIZATION AND PATCHING

Given a context window $x = (x_1, \ldots, x_L) \in \mathbb{R}^L$, we first standardize it using statistics computed over the context itself,

$$\tilde{x}_t = \frac{x_t - \mu}{\sigma + \epsilon},$$

making tokenization invariant to offline shifts in mean and magnitude. The normalized signal is divided into $N = \lfloor L/P \rfloor$ non-overlapping patches of length P, $p^{(1)}, \ldots, (p^{(N)}$ with $p^{(i)}) \in \mathbb{R}^P$. Each patch also carries an observation mask, so that patches with missing values are handled without discarding the surrounding context.

A.1.2 DISCRETE PATCH IDENTIFIERS

To enable BPE motif discovery, we summarize each patch with a discrete identifier. We use a pre-learned codebook of C centers $\{m_1, \ldots, m_C\} \in \mathbb{R}^P$ via k-means clustering over normalized patches from a training corpus. At tokenization time, each patch $p^{(i)}$ is mapped to its nearest center,

$$e_i = \arg \min_{k \in \{1,\ldots,C\}} \left\| o^{(i)} \odot \left(p^{(i)} - m_k\right) \right\|^2,$$

where distance is computed over unmasked positions using the observation mask $o^{(i)} \in \{0, 1\}^P$. This yields a sequence of patch identifiers $e = (e_1, \ldots, e_N)$ that captures local structural shapes across the signal.

A.1.3 BPE MOTIF MERGING

We apply Byte-Pair Encoding (BPE) over e to iteratively group frequently co-occurring adjacent patches into variable-length *motifs*. During offline tokenization training, BPE repeatedly merges the most frequent adjacent pair across a training corpus, recording the occurrence count k associated with each learned merge rule. At inference time, rather than applying the entire rule set, we introduce a test-time threshold $k_{\min}$. BPE applies only the merge rules satisfying $k \geq k_{\min}$. Lowering $k_{\min}$ admits less frequent rules and therefore enables more aggressive compression. Additionally, to prevent excessive compression and preserve temporal resolution, we enforce a maximum motif length $L_{\max}$. This process segments the N base patches into $M \leq N$ contiguous motifs, while offering a flexible control over sequence length.

A.2 REPRESENTATION STAGE

To eliminate quantization error, we use the discovery stage strictly to establish motif boundaries. Consequently, we must map variable-length groups of continuous patches within each motif to a fixed-dimensional token embedding. Simple strategies such as zero-padding to a maximum length performed poorly in our preliminary experiments, as the forecasting model was unable to leverage these representations effectively. Inspired by Abeywickrama et al. (2025), we employ an adaptive embedding module that combines within-motif self-attention, pooling, and cross-attention as shown in Figure 1b. Importantly, because these operations are computed across the patches *within* each motif rather than across the whole sequence length, their computational overhead scales linearly with respect to sequence length. Our embedding pipeline consists of three distinct steps:

A.2.1 PATCH ENCODING & SELF-ATTENTION

For a given motif $\mathcal{M}_j$ spanning base patches from start index a to end index b, each patch $p^{(i)}$ and its observation mask $o^{(i)}$ are projected into an embedding space via a residual Multi-Layer Perceptron

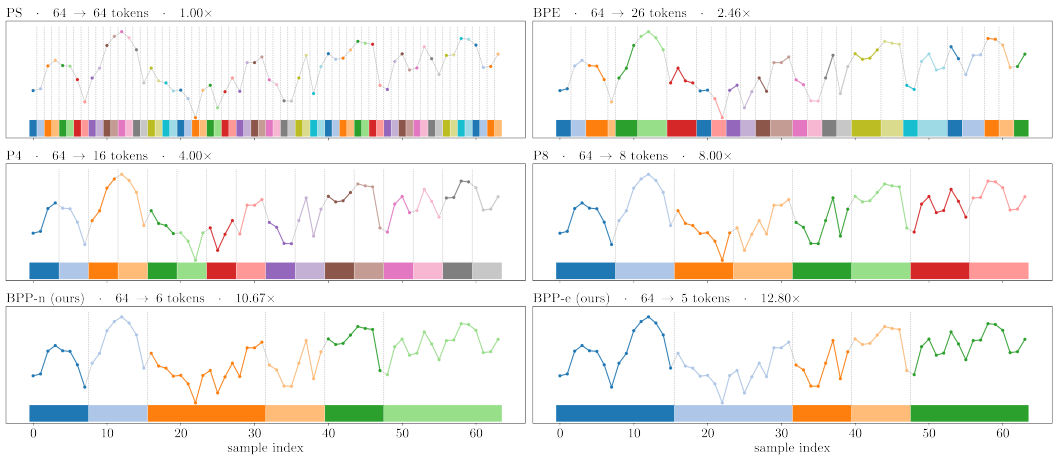


Figure 4: Tokenization of the same 64-observation window using different methods. Colored segments indicate token boundaries, and each panel reports the resulting token count and compression.

(MLP) and augmented with a global positional encoding reflecting the patch's absolute position in the full sequence. We then apply self-attention strictly across the patches within the motif boundaries, yielding $\{h_i\}_{i=a}^{b} \in \mathbb{R}^d$:

$$\tilde{h}_i = \mathrm{MLP}\big([p^{(i)}\,;\,o^{(i)}]\big) + \mathrm{PE}(i), \qquad \{h_i\}_{i=a}^{b} = \mathrm{SelfAttn}\left(\{\tilde{h}_i\}_{i=a}^{b}\right).$$

A.2.2 Pooling

Next, the patch representations within motif $\mathcal{M}_j$ are pooled into a single vector u_j. In our experiments, extracting the representation of the motif's *last* patch yielded the best performance, which is what we use in BPP. To retain temporal metadata, we construct a query vector q_j by adding a learned length embedding $\mathrm{LE}(|\mathcal{M}_j|)$ and the global positional encoding of the motif's final patch $\mathrm{PE}(b)$:

$$u_j = h_b, \qquad q_j = u_j + \mathrm{LE}(|\mathcal{M}_j|) + \mathrm{PE}(b).$$

A.2.3 Cross-Attention

Finally, the query q_j attends over the features $\{h_i\}_{i=a}^{b}$ of its constituent patches through a cross-attention layer. This produces the final fixed-dimensional motif token $z_j \in \mathbb{R}^d$:

$$z_j = \mathrm{CrossAttn}\Big(\mathrm{Query} = q_j,\ \mathrm{Keys} = \{h_i\}_{i=a}^{b},\ \mathrm{Values} = \{h_i\}_{i=a}^{b}\Big),$$

A.3 Test-time Compression Control

As established in A.1.3, BPP provides a training-free knob to control compression during inference. Beyond the $P\times$ compression gained from patching, BPE motif merging can be controlled by filtering merge rules based on occurrence counts. Specifically, given a user-defined threshold $k_{\min}$, tokenization applies only merge rules with occurrence count $k \geq k_{\min}$.

Increasing $k_{\min}$ retains only the most common merges, yielding shorter motifs and more tokens. Conversely, lowering $k_{\min}$ allows more merges and compresses more aggressively. Because the full merge vocabulary is learned once and our representation stage dynamically processes motifs up to $L_{\max}$, adjusting $k_{\min}$ requires no retraining of either the tokenizer or the backbone.

This provides a single dial to adapt sequence length to computational constraints. In our experiments, we evaluate **BPP-n** with $k_{\min} = 10{,}000$ and the more compressed **BPP-e** with $k_{\min} = 1{,}000$, using the same checkpoint trained with $k_{\min} = 20{,}000$. Figure 4 compares the tokenizations produced by our two operating points with those of the baselines.

Table 3: Fixed-window forecasting MSE across 20 datasets and three model scales (lower is better). Bold and underline denote the best and second-best results within each dataset and scale, respectively.

	Tiny							Small							Base						
Dataset	PS	P4	P8	P16	BPE	BPP-n	BPP-e	PS	P4	P8	P16	BPE	BPP-n	BPP-e	PS	P4	P8	P16	BPE	BPP-n	BPP-e
ETTh1	0.467	0.407	0.410	0.416	0.471	**0.405**	0.406	0.450	0.398	0.401	0.398	0.433	**0.397**	0.399	0.426	0.397	0.397	0.399	0.412	**0.396**	0.398
ETTh2	0.224	0.211	0.210	0.208	0.222	**0.204**	0.206	0.211	**0.203**	**0.203**	0.205	0.214	**0.203**	0.206	0.205	0.203	**0.198**	0.202	0.210	0.200	0.202
ETTm1	0.382	0.358	0.349	0.340	0.415	**0.331**	0.338	0.362	0.335	0.322	**0.320**	0.378	0.323	0.329	0.388	0.327	0.329	**0.326**	0.359	**0.326**	0.331
ETTm2	0.156	0.150	0.147	0.145	0.158	**0.142**	**0.142**	0.149	0.147	0.144	0.144	0.153	**0.142**	0.143	0.148	0.146	0.145	0.143	0.150	**0.142**	**0.142**
Weather	**0.142**	0.148	0.151	0.150	0.155	0.157	0.157	**0.139**	0.146	0.147	0.145	0.148	0.148	0.149	**0.143**	0.148	0.148	0.145	0.147	0.146	0.147
Exchange	**0.121**	0.130	0.132	0.135	0.132	0.127	0.127	**0.117**	0.130	0.131	0.132	0.132	0.130	0.131	**0.116**	0.137	0.131	0.136	0.127	0.129	0.130
PEMS03	**0.258**	0.291	0.308	0.338	0.267	0.315	0.304	0.344	0.307	0.302	**0.252**	0.266	0.256	0.263	0.328	0.292	0.257	**0.223**	0.249	0.231	0.224
PEMS04	**0.591**	0.605	0.618	0.638	0.600	0.618	0.611	0.646	0.615	0.610	0.588	0.588	**0.582**	0.586	0.626	0.594	0.586	0.568	0.581	0.559	**0.551**
Saugeenday D	0.923	0.888	0.843	0.821	1.035	0.802	**0.800**	0.867	0.832	0.806	**0.797**	0.963	0.816	0.821	0.833	0.780	**0.777**	0.792	0.865	0.805	0.808
Saugeenday M	0.535	0.423	0.414	**0.405**	0.634	0.411	0.411	0.452	0.417	0.437	**0.384**	0.583	0.427	0.427	**0.404**	0.424	0.411	0.405	0.458	0.418	0.430
Saugeenday W	0.749	0.737	0.732	**0.668**	0.796	0.671	0.670	0.740	0.676	0.671	**0.654**	0.774	0.674	0.674	0.709	0.677	0.666	0.652	0.696	**0.615**	0.641
Bizitobs l2c H	0.293	**0.225**	0.280	0.290	0.361	0.314	0.289	0.325	0.328	**0.267**	0.270	0.323	0.325	0.312	0.357	0.325	0.371	0.341	0.326	0.294	**0.270**
Bizitobs l2c 5T	**0.103**	0.125	0.127	0.131	0.150	0.129	0.130	**0.120**	0.123	0.125	0.126	0.134	0.123	0.124	**0.118**	0.147	0.131	0.135	0.127	0.132	0.131
Bizitobs App	0.019	0.008	0.007	0.010	0.014	**0.006**	0.007	0.012	0.013	0.017	0.012	0.013	0.010	**0.009**	0.009	0.012	0.017	0.014	**0.008**	0.012	0.012
Bizitobs S	0.010	0.010	0.007	0.008	0.010	**0.006**	**0.006**	**0.007**	0.009	0.011	0.009	0.010	**0.007**	**0.007**	**0.006**	0.010	0.011	0.011	0.007	0.009	0.009
Solar D	**0.581**	0.662	0.649	0.665	0.597	0.634	0.627	**0.563**	0.659	0.652	0.670	0.639	0.671	0.678	**0.572**	0.662	0.689	0.664	0.618	0.677	0.674
Solar H	0.268	**0.251**	0.253	0.253	0.290	0.263	0.275	0.252	0.257	0.246	**0.245**	0.264	0.259	0.274	0.249	**0.231**	0.238	0.238	0.260	0.249	0.262
m4 H	**0.001**	0.002	0.002	**0.001**	0.002	0.002	0.002	**0.002**	**0.002**	**0.002**	**0.002**	0.003	**0.002**	0.003	**0.001**	**0.001**	**0.001**	**0.001**	0.003	**0.001**	0.002
m4 D	**0.045**	0.057	0.055	0.055	0.061	0.055	0.054	**0.045**	0.058	0.058	0.057	0.056	0.056	0.056	**0.044**	0.059	0.058	0.059	0.054	0.058	0.058
m4 W	**0.065**	0.099	0.094	0.099	0.110	0.095	0.094	**0.059**	0.096	0.096	0.101	0.107	0.094	0.094	**0.065**	0.096	0.100	0.101	0.098	0.096	0.096
Avg.	0.297	0.289	0.289	0.289	0.324	0.284	**0.283**	0.293	0.288	0.282	**0.276**	0.309	0.282	0.284	0.287	0.283	0.283	0.278	0.288	**0.275**	0.276

B EXPERIMENTS

B.1 SETUP

B.1.1 BASELINES

Shared preprocessing. Unless a method prescribes its own preprocessing, we apply the same z-score normalization used by BPP. During training, the statistics are computed over the observed values of the entire available series. At inference, they are computed only over the observed context. All tokenizers are integrated with the same forecasting backbone and training pipeline to isolate the effect of tokenization.

Point-wise tokenization. For **PS**, we use the tokenizer implementation from the official Chronos repository (Ansari et al., 2024). We retain its original mean-scaling procedure, including its minimum-scale safeguard, rather than applying our shared z-score normalization. Each scaled observation is quantized into a discrete token using the Chronos uniform-bin tokenizer.

Fixed patching. We implement **P4**, **P8**, and **P16** within our framework. Each method divides the input into non-overlapping patches containing 4, 8, or 16 observations, respectively, and maps each patch to a fixed-dimensional token (Nie et al., 2022).

LightGTS. We implement **LightGTS** using its officially released code (Wang et al., 2025). The method estimates the dominant period of each input using the Fourier transform and uses the resulting period as its patch length. Its Flex Projection layer maps patches of different lengths into a shared embedding space. The original configuration sets `canonical_patch_len` to 24 and permits patch lengths up to 96. For **LightGTS-16**, we cap the patch length at 16 and reduce `canonical_patch_len` from 24 to 8 to accommodate the smaller patch-length range.

TimeSqueeze. Because no public implementation of **TimeSqueeze** was available, we implement its tokenization procedure following Ankireddy et al. (2026). A Mamba encoder first processes the signal at full resolution. Patch boundaries are then placed when the difference between consecutive observations exceeds a threshold determined by the local signal power, and the Mamba representations at these boundaries are retained as tokens. We use a boundary threshold of 0.3, a power lookback window of 8, and a maximum patch length of 8. The original method uses two Mamba layers, but this configuration was unstable and kept producing NaNs during our training runs. We therefore use one Mamba layer. To keep the parameter count comparable to the other methods, we use a Mamba hidden dimension of 192, state dimension of 64, convolution width of 4, and expansion factor of 2.

Time-series BPE. Because no public implementation was available, we implement **BPE** following Götz et al. (2025). We use the paper's medium-compression configuration, which uniformly quantizes normalized observations into 37 bins over the range $[-5, 5]$ and retains motifs occurring at least $1{,}000$ times. BPE is then applied directly to the resulting discrete sequence to construct the motif vocabulary.

BPP-n/BPP-e (ours). Both configurations use the same tokenizer vocabulary and forecasting checkpoint. During training, merge rules with occurrence counts of at least $k_{\min} = 20{,}000$ are applied. At inference, BPP-n uses $k_{\min} = 10{,}000$, while BPP-e lowers the threshold to $1{,}000$, admitting less frequent merges and increasing compression.

BPP-n-q/BPP-e-q (ours, quantized) are ablations that evaluate the benefit of preserving continuous signal values. For each patch $p^{(i)}$, we replace it with its assigned K-means center before passing it to the representation stage. The motif boundaries, compression thresholds, architecture, and training protocol remain unchanged from the corresponding BPP-n and BPP-e configurations. We evaluate these variants at the base model scale.

B.1.2 Model and Training Protocol

Tokenizer training. Following Götz et al. (2025), we use the same random sample of $100{,}000$ univariate series from the Chronos pretraining corpus (Ansari et al., 2024) to train both BPE and BPP. For BPP, we first fit the K-means codebook, convert the patches from these series into discrete IDs, and then learn the BPE merge vocabulary from the resulting sequences. The BPP and BPE tokenizers are trained once and reused across all model scales. LightGTS and TimeSqueeze contain trainable representation components and are therefore trained separately with each model scale.

Forecasting-model training. We use T5 encoders at three scales: tiny, small, and base followed by a MLP regression head. All models are trained on the full Chronos pretraining corpus for $200{,}000$ steps using contexts length of 512 and a global batch size of 256, distributed across two GPUs with 128 samples per GPU. All methods had converged by the end of training. The encoder-only backbone predicts the next 64 observations using an MLP regression head and is optimized with MSE. We use AdamW with a default learning rate of 3×10^{-3} and a warm-up ratio of 0.01. When this learning rate caused numerical instability, we reduced it for the affected method. All remaining optimization and scheduling settings follow the Chronos training pipeline (Ansari et al., 2024).

Randomness and compute. The main results are obtained from a single training run because repeating pretraining across every method and model scale is computationally expensive. We additionally repeat representative configurations with multiple random seeds in Table 4, where we observe only minor variation. Depending on availability, models are trained using two NVIDIA RTX 6000, A10, or A100 GPUs. Each evaluation uses one GPU, and all latency measurements are performed on the same RTX 6000 GPU.

B.2 Results

In this section, we discuss and present the evaluations of BPP.

B.2.1 GIFT-Eval

Table 1 reports forecasting performance and average compression on GIFT-Eval. PS provides no compression, while BPE achieves only $4.08\times$ compression and consistently underperforms P4 at a comparable $4\times$ compression rate. In contrast, BPP-n achieves the lowest relative MSE at the small and base scales and remains close to P16 at the tiny scale while compressing inputs by $15.10\times$. BPP-e increases compression to $18.54\times$, exceeding P16, with a modest reduction in accuracy. Since both operating points share a checkpoint, these results demonstrate that BPP can adjust its accuracy–compression trade-off at inference time without retraining.

Table 4: Fixed-window forecasting MSE for base-scale models across seeds 7, 23, and 42. Values are mean ± sample standard deviation (lower is better). Bold and underline indicate the best and second-best unrounded means per dataset. "Avg." is the unweighted average across datasets, and "Comp." is the average compression factor.

Dataset	PS	P4	P8	P16	BPE	BPP-n	BPP-e
ETTh1	0.4341 ± 0.0074	$\mathbf{0.3943} \pm 0.0021$	$\underline{0.3949} \pm 0.0014$	0.3970 ± 0.0019	0.4184 ± 0.0054	0.3969 ± 0.0008	0.3990 ± 0.0020
ETTh2	0.2101 ± 0.0061	$\underline{0.2011} \pm 0.0023$	$\mathbf{0.1984} \pm 0.0005$	0.2011 ± 0.0016	0.2105 ± 0.0015	0.2015 ± 0.0010	0.2030 ± 0.0013
ETTm1	0.3921 ± 0.0049	0.3263 ± 0.0017	$\mathbf{0.3251} \pm 0.0034$	$\underline{0.3257} \pm 0.0017$	0.3615 ± 0.0027	0.3276 ± 0.0024	0.3344 ± 0.0040
ETTm2	0.1495 ± 0.0023	0.1454 ± 0.0006	0.1445 ± 0.0011	0.1439 ± 0.0005	0.1505 ± 0.0001	$\underline{0.1434} \pm 0.0008$	$\mathbf{0.1431} \pm 0.0012$
Weather	$\mathbf{0.1431} \pm 0.0008$	0.1474 ± 0.0007	0.1478 ± 0.0002	$\underline{0.1462} \pm 0.0007$	0.1466 ± 0.0009	0.1475 ± 0.0018	0.1478 ± 0.0017
Exchange	$\mathbf{0.1149} \pm 0.0018$	0.1330 ± 0.0036	0.1312 ± 0.0011	0.1352 ± 0.0006	0.1299 ± 0.0021	0.1298 ± 0.0006	$\underline{0.1295} \pm 0.0004$
PEMS03	0.3528 ± 0.0275	0.2892 ± 0.0199	0.2606 ± 0.0042	0.2362 ± 0.0144	0.2496 ± 0.0046	$\underline{0.2359} \pm 0.0063$	$\mathbf{0.2328} \pm 0.0085$
PEMS04	0.6454 ± 0.0172	0.5944 ± 0.0023	0.5844 ± 0.0022	0.5749 ± 0.0077	0.5827 ± 0.0036	$\underline{0.5697} \pm 0.0094$	$\mathbf{0.5627} \pm 0.0100$
SaugeenDay D	$\mathbf{0.7799} \pm 0.0463$	$\underline{0.7823} \pm 0.0307$	0.8006 ± 0.0253	0.7940 ± 0.0055	0.9222 ± 0.0667	0.8027 ± 0.0068	0.8052 ± 0.0121
SaugeenDay M	0.4325 ± 0.0244	0.4267 ± 0.0053	$\mathbf{0.4075} \pm 0.0161$	$\underline{0.4119} \pm 0.0134$	0.4698 ± 0.0173	0.4168 ± 0.0065	0.4254 ± 0.0049
SaugeenDay W	0.6902 ± 0.0159	0.6847 ± 0.0076	0.6691 ± 0.0071	0.6556 ± 0.0030	0.6940 ± 0.0113	$\mathbf{0.6268} \pm 0.0116$	$\underline{0.6439} \pm 0.0062$
BizITObs L2C H	0.4113 ± 0.0480	0.3392 ± 0.0216	0.3577 ± 0.0119	$\underline{0.3153} \pm 0.0230$	0.3177 ± 0.0086	0.3169 ± 0.0265	$\mathbf{0.2897} \pm 0.0215$
BizITObs L2C 5T	$\mathbf{0.1214} \pm 0.0033$	0.1459 ± 0.0026	0.1330 ± 0.0021	0.1350 ± 0.0010	0.1295 ± 0.0022	0.1290 ± 0.0027	$\underline{0.1273} \pm 0.0028$
BizITObs App	0.0117 ± 0.0028	0.0134 ± 0.0018	0.0159 ± 0.0015	0.0148 ± 0.0016	$\mathbf{0.0083} \pm 0.0019$	0.0103 ± 0.0025	$\underline{0.0103} \pm 0.0022$
BizITObs S	$\mathbf{0.0074} \pm 0.0014$	0.0097 ± 0.0006	0.0106 ± 0.0007	0.0105 ± 0.0004	0.0079 ± 0.0013	$\underline{0.0076} \pm 0.0014$	0.0077 ± 0.0014
Solar D	$\mathbf{0.5561} \pm 0.0162$	0.6640 ± 0.0073	0.6729 ± 0.0316	0.6475 ± 0.0162	$\underline{0.6405} \pm 0.0384$	0.6674 ± 0.0079	0.6691 ± 0.0065
Solar H	0.2487 ± 0.0038	$\mathbf{0.2359} \pm 0.0079$	$\underline{0.2403} \pm 0.0028$	0.2411 ± 0.0035	0.2595 ± 0.0008	0.2510 ± 0.0019	0.2708 ± 0.0075
M4 H	0.0015 ± 0.0001	0.0011 ± 0.0002	$\underline{0.0010} \pm 0.0002$	$\mathbf{0.0010} \pm 0.0001$	0.0024 ± 0.0003	0.0014 ± 0.0003	0.0016 ± 0.0003
M4 D	$\mathbf{0.0435} \pm 0.0004$	0.0591 ± 0.0008	0.0592 ± 0.0011	0.0587 ± 0.0005	$\underline{0.0555} \pm 0.0016$	0.0580 ± 0.0005	0.0574 ± 0.0008
M4 W	$\mathbf{0.0641} \pm 0.0022$	0.1042 ± 0.0108	0.0995 ± 0.0037	0.1017 ± 0.0049	0.0997 ± 0.0019	0.0958 ± 0.0009	$\underline{0.0956} \pm 0.0009$
Avg.	0.2905 ± 0.0028	0.2849 ± 0.0028	0.2827 ± 0.0004	$\underline{0.2774} \pm 0.0020$	0.2928 ± 0.0050	$\mathbf{0.2768} \pm 0.0021$	0.2778 ± 0.0025
Avg. rank	4.000	4.400	4.250	3.650	4.850	**3.300**	$\underline{3.550}$
Comp. (×)	1.000	4.000	8.000	16.000	4.753	$\underline{16.919}$	**20.542**

B.2.2 Fixed-Window Evaluation and Robustness

We further evaluate the trained checkpoints on fixed-window forecasting on 20 datasets widely used in the literature. Following Götz et al. (2025), we obtain the ETT family comprising ETTh1, ETTh2, ETTm1, and ETTm2 (Zhou et al., 2021), together with Weather (Wu et al., 2021), from the Chronos evaluation pipeline (Ansari et al., 2024). We manually add PEMS03 and PEMS04 (Song et al., 2020) and Exchange (Lai et al., 2018), extracting evaluation windows with a stride of 32. The remaining configurations are drawn from the GIFT-Eval test split (Aksu et al., 2024) and include SaugeenDay (Godahewa et al., 2021), BizITObs, Solar (Lai et al., 2018), and M4.

Table 3 and Table 5 illustrate the results across datasets. BPP-e achieves the lowest average error at the tiny scale with $20.54\times$ compression, while BPP-n achieves the lowest error at the base scale with $16.92\times$ compression. At the small scale, BPP-n ranks second behind P16 while providing slightly greater compression. Overall, the two BPP configurations occupy the best and second-best positions at the tiny and base scales and remain competitive at the small scale. In contrast, point-wise BPE provides only $4.75\times$ compression and yields the highest average error at the tiny and small scales.

To compare the accuracy–compression trade-off across datasets, we normalize each model's forecasting error and compression relative to P8 and then average the resulting ratios across the 20 datasets. Under this normalization, P8 lies at $(1, 1)$, lower values on the vertical axis indicate better forecasting, and higher values on the horizontal axis indicate greater compression. Figure 2a visualizes the results for the base-scale models. Both BPP configurations lie in the Pareto frontier in the favorable lower-right region. BPP-n achieves slightly better average accuracy than P8 with approximately $2.1\times$ its compression, while BPP-e increases this to approximately $2.6\times$ with only a small accuracy trade-off. Both also provide greater compression than P16 while maintaining comparable forecasting performance.

B.2.3 Effect of Seeding

To assess robustness to the random seed, we repeat training of the base-scale models with seeds 7 and 23, in addition to the original seed 42, and evaluate all checkpoints in the fixed-window setting. Table 4 reports the mean and standard deviation across the three seeds, showing that our findings remain consistent across runs.

B.2.4 Inference efficiency experiment

Average compression does not fully capture the computational savings during batched inference. Because BPP produces sequences of different lengths, the sequences within each batch must be

Table 5: Compression factors across the 20 fixed-window datasets (higher is better). Compression is independent of model scale. Bold and underline indicate the best and second-best results per dataset.

Dataset	PS	P4	P8	P16	BPE	BPP-n	BPP-e
ETTh1	1.000	4.000	8.000	<u>16.000</u>	2.815	13.005	**16.254**
ETTh2	1.000	4.000	8.000	<u>16.000</u>	2.945	12.785	**16.085**
ETTm1	1.000	4.000	8.000	<u>16.000</u>	3.458	15.550	**19.307**
ETTm2	1.000	4.000	8.000	<u>16.000</u>	3.552	14.643	**18.124**
Weather	1.000	4.000	8.000	16.000	6.356	<u>19.684</u>	**23.148**
Exchange	1.000	4.000	8.000	16.000	4.891	<u>19.200</u>	**23.190**
PEMS03	1.000	4.000	8.000	16.000	3.733	<u>19.405</u>	**23.250**
PEMS04	1.000	4.000	8.000	16.000	3.713	<u>18.377</u>	**22.081**
Saugeenday D	1.000	4.000	8.000	16.000	7.859	<u>17.564</u>	**20.521**
Saugeenday M	1.000	4.000	8.000	**16.000**	2.821	10.139	<u>13.474</u>
Saugeenday W	1.000	4.000	8.000	<u>16.000</u>	3.854	12.800	**17.162**
Bizitobs l2c H	1.000	4.000	8.000	16.000	5.769	<u>17.198</u>	**20.789**
Bizitobs l2c 5T	1.000	4.000	8.000	16.000	9.118	<u>22.662</u>	**25.545**
Bizitobs App	1.000	4.000	8.000	16.000	7.783	<u>24.548</u>	**26.018**
Bizitobs S	1.000	4.000	8.000	16.000	6.783	<u>22.081</u>	**24.772**
Solar D	1.000	4.000	8.000	**16.000**	2.136	10.745	<u>13.873</u>
Solar H	1.000	4.000	8.000	<u>16.000</u>	4.775	15.992	**22.532**
M4 H	1.000	4.000	8.000	<u>16.000</u>	3.238	14.340	**19.594**
M4 D	1.000	4.000	8.000	16.000	4.575	<u>18.693</u>	**22.620**
M4 W	1.000	4.000	8.000	16.000	4.875	<u>18.976</u>	**22.494**
Mean	1.000	4.000	8.000	16.000	4.753	<u>16.919</u>	**20.542**

padded to the length of the longest sequence, potentially reducing the effective compression. We therefore directly measure model-side inference latency using the base-scale models and a fixed batch of 64 context windows from SaugeenDay/D, each containing 512 observations. For each tokenizer, we prepare and pad the tokenized inputs before timing and measure the forward pass through its corresponding input layer, the T5 encoder, and the regression head, which predicts the next 64 observations.

All models are compiled using `torch.compile`. We perform five warm-up runs followed by 20 timed runs and report per-sample latency by dividing the average batch latency by 64. Tokenization is excluded to isolate the computational effect of sequence-length reduction within the neural model and avoid conflating it with implementation-specific preprocessing overhead. Accordingly, these measurements reflect model-side rather than end-to-end inference efficiency. Figure 2b presents the resulting per-sample latencies. Under this setting, BPP reduces inference latency by up to 25% relative to P16, demonstrating that its higher compression translates into practical computational savings even after accounting for batch padding.

B.2.5 Representation Stage Ablations

We ablate the main components of the representation stage to determine how each contributes to forecasting performance. All variants use the same tokenizer as BPP, and each is trained from scratch at the base model scale and evaluated on GIFT-Eval. We consider three modifications. First, *Max Pooling* replaces the final-patch representation with element-wise max pooling over the contextualized patches. Second, *No Self-Attention* removes within-motif self-attention while retaining the remaining pooling and cross-attention operations. Third, *Padded MLP* replaces the entire representation stage with an MLP. For this variant, each motif is padded to $L_{\max}$ patches and flattened into a vector containing the continuous values, observation masks, and motif length. The resulting input has dimension $2PL_{\max} + 1$. Comparing these variants with the original architecture isolates the effects of the pooling strategy, within-motif contextualization, and adaptive representation module.

B.3 Compute Resources

Each training run used two NVIDIA RTX PRO 6000 GPUs with 96 GB of memory each. Training required approximately 2–14 hours, depending on the model scale and tokenizer compression, with PS combined with the base model size requiring the longest runtime. Evaluations used a single GPU of the same type and required approximately 1–2 hours per checkpoint for fixed-window forecasting and one hour for GIFT-Eval. The same hardware is also used for the latency experiments.