Dynamic Tokenization via Reinforcement Patching: End-to-end Training and Zero-shot Transfer

Yulun Wu † ¶ * **Sravan Kumar Ankireddy** † § * **Samuel Sharpe** ¶

Nikita Seleznev ¶ **Dehao Yuan** ¶ **Hyeji Kim** § **Nam H. Nguyen** ¶

Abstract

Efficiently aggregating spatial or temporal horizons to acquire compact representations has become a unifying principle in modern deep learning models, yet learning data-adaptive representations for long-horizon sequence data, especially continuous sequences like time series, remains an open challenge. While fixed-size patching has improved scalability and performance, discovering variable-sized, data-driven patches end-to-end often forces models to rely on soft discretization, specific backbones, or heuristic rules. In this work, we propose Reinforcement Patching (ReinPatch), a novel framework that jointly optimizes a sequence patching policy and its downstream sequence backbone model with reinforcement learning. By formulating patch boundary placement as a discrete decision process optimized via Group Relative Policy Gradient (GRPG), ReinPatch bypasses the need for continuous relaxations and performs dynamic patching policy optimization in a natural manner. Moreover, our method allows strict enforcement of a desired compression rate, freeing the downstream backbone to scale efficiently, and natively supports multi-level hierarchical modeling. We evaluate ReinPatch on time-series forecasting datasets, where it demonstrates compelling performance compared to state-of-the-art data-driven patching strategies. Furthermore, our detached design allows the patching module to be extracted as a standalone foundation patcher, providing the community with visual and empirical insights into the segmentation behaviors preferred by a purely performance-driven neural patching strategy.

1 Introduction

A unifying principle in modern deep learning across domains is the efficient transformation of raw signals into rich representations, compressing spatial or temporal horizons into hidden dimensions through either architectural design or strategic pre-processing. In language modeling, algorithms such as byte-pair encoding (BPE) [23] have become a standard pre-processing technique, condensing character-level inputs into compact, semantically meaningful sub-words that are subsequently projected into continuous embedding spaces. Similarly, in computer vision, segmenting an image into non-overlapping patches and projecting each patch independently has emerged as a canonical architectural choice, forming the basis of powerful Vision Transformers (ViT) [6]. The shared success of these tokenization strategies highlights a broader principle: how an input signal is represented is often as important as the choice of backbone architecture itself.

Although fixed-size patching and downsampling within convolutional neural network [12] blocks are standard strategies for embedding compression in vision models, the same does not always hold true

† Correspondence to: <yulun.wu@capitalone.com, sravan.ankireddy@utexas.edu>

¶ Capital One, § University of Texas, Austin

* Equal Contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Table 1: Qualitative comparison of ReinPatch with existing patching strategies across key design properties. **Characteristics: Adaptive** (Data-adaptive); **Var. Size** (Variable-sized); **Model** (Model-based); **Heuristic-free** (Unreliant on human-designed heuristics or rules); **Agnostic** (Unrestricted to particular data types); **Hierarchical** (Supports hierarchical patching). **Optimization: E2E** (End-to-end); **Non-local** (Unlimited to local exploration during end-to-end training); **Detached** (Detached from the downstream sequence model). ✓: satisfied, ✗: not satisfied, ✓/✗: partially satisfied, n/a: not applicable.

	Characteristics						Optimization		
Method	**Adaptive**	**Var. Size**	**Model**	**Heuristic-free**	**Agnostic**	**Hierarchical**	**E2E**	**Non-local**	**Detached**
Static Patching [18]	✗	✗	✗	✓	✓	✓	✗	n/a	✓
Entropy Patching [19]	✓	✓	✓	✗	✗	✗	✗	n/a	✓
TimeSqueeze's Compression [2]	✓	✓	✗	✗	✗	✗	✗	n/a	✓
H-Net's Chunking [10]	✓	✓	✓	✗	✓	✓	✓	✗	✗
Periodic Patching [26]	✓	✗	✗	✗	✗	✗	✗	n/a	✓
Selective Patching [29]	✓	✗	✓	✗	✓	✗	✓/✗	✗	✗
ReinPatch (Ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓

for sequence data such as language and time series. Instead, the inductive bias of variable-sized, data-adaptive segmentation strategies may often align better with the inherent nature of these sequences. For instance, the most prevalent tokenization approach in sequence applications such as language has been compression via pre-processing (i.e. BPE tokenization) instead of via neural modeling. However, algorithms designed for discrete sequence tokenization such as BPE are not naturally well-suited for other sequences such as time series due to the continuous nature of these data. Therefore, many early transformer-based approaches in time series [32, 28, 33] did not have a compression component and projected each individual time step independently into the embedding space, producing prohibitively long sequences that scale poorly because of the quadratic complexity of self-attention. More recently, PatchTST [18] experimented with fixed-size, non-data-adaptive patching on time series, introducing a simple but effective mechanism that groups a fixed number of contiguous time steps into a *patch* and represents each patch with a single embedding, thereby compressing the input signal and shortening the sequence length. Since then, some recent work have explored data-driven strategies for selecting patch sizes or boundary locations to replace static patching, either by combining existing patches to form larger patches and richer representations [9, 29] or by relying on auxiliary external rules [2, 26] or external models [1]. Even for language models, more recent works such as BLT [19] and H-Net [10] have also investigated using various patching strategies (also known as *chunking*) as part of the neural model design to replace BPE tokenizer.

While much of mainstream deep learning research has focused on language and vision, time-series data remains central to many high-impact domains, including energy, finance, climate, and healthcare. Furthermore, it is a domain mostly consisting of long-horizon continuous data where patching strategies are crucial to methodology design. The simple static patching approach in PatchTST alone led to strong forecasting gains, demonstrating that efficient compression is a critical factor even when the backbone architecture remains unchanged.

In this work, we further explore the end-to-end optimization of patching strategy alongside the training of deep sequence models. In contrast to prior works such as H-Net [10] and SRSNet [29], which incorporate a patching component into the neural design of their main backbone model and rely on continuous relaxations of patching boundary placement, we look at this problem through a different angle and use reinforcement learning to naturally and canonically solve what is essentially a discrete action optimization problem. This means we need not rely on soft discretization of patching boundaries to enable gradient flow, and we allow the patching policy to be independent of the neural design of the backbone model. We elaborate the benefits of doing so throughout Section 2 and Section 3, and provide a summary in advance in Table 1 to give the reader an overview of the advantages of our approach. Our contributions are as follows:

- We propose *Reinforcement Patching*, or *ReinPatch* for short, which to our knowledge is the first framework to use reinforcement learning for joint optimization of a deep sequence model and a sequence patching policy in an end-to-end fashion while keeping the patching policy as a separate neural module and treating the deep sequence model as its *environment*.
- This separation allows a standalone foundation patcher to be established after large-scale joint pre-training, and allows continued pre-training of the deep sequence model with the

patcher's weights frozen. As a proof of concept, we produce a foundation patcher on univariate time series and benchmark its zero-shot performance in our experiments.

- We focus on time-series forecasting in experiments and show that our method improves over a broad set of fixed- and variable-size patching strategies under a unified hierarchical backbone on multiple forecasting datasets. We perform analysis and ablation studies to help readers understand the behavior of ReinPatch, and visualize the patches it generates to provide insight into the segmentation patterns preferred by a performance-driven neural patching model.

2 Related Work

Recent advancements in sequence modeling have heavily relied on static patching methods, popularized by architectures like PatchTST [18], which use uniform patch sizes to improve training and inference efficiency. This design has been adopted by various time-series foundation models, including TimesFM [3], MOMENT [8], Moirai [27], and Timer-XL [14]. However, static patching struggles to capture substantial temporal heterogeneity within real-world series. To address this, several data-adaptive strategies have emerged, such as HDMixer [9], LightGTS [26], and SRSNet [29], which automatically adjust patch sizes or combine fixed-size patches. Yet, these methods do not explicitly learn dynamic, within-sequence policies based on local signal characteristics.

Other approaches rely on external rules or auxiliary models to determine boundaries, such as BLT and its variants [19, 1, 11], which use a separately trained entropy model, SpaceByte [25], which uses natural word separators, or TimeSqueeze [2], which uses local variance. Because these depend on modality-specific heuristics or external identifiers, they cannot be jointly optimized with the main sequence backbone. Alternatively, end-to-end patching methods, like H-Net [10], enable gradient-based patch learning through smoothed routing, but this tightly couples the patching strategy to the main model's training. This coupling introduces several limitations, such as preventing a regime where the patching module is pre-trained and frozen, and restricting the approach strictly to causal modeling due to the architectural design of the main model. Besides, H-Net has a few other drawbacks compared to our proposed method in terms of policy exploration, model pre-planning, and inductive bias heuristics. We will further elaborate on these topics in the following section. A more extended discussion of related work can be found in Appendix A.

3 Proposed Method

In this section, we describe the formulation, optimization, and adaptation of our proposed method in details. For an overview of the method, see Figure 1.

3.1 Modeling Framework

Let $\mathbf{x}_T = x_{0:T}$ be a sequence data with initial point x_0 and T subsequent evolutions $\{x_1, \ldots, x_T\}$. Let $\mathbf{b}_T = b_{-1:T}$ be an auxiliary sequence of $\mathbf{x}_T$ where $b_t \in \{0, 1\}$ for $0 \leq t \leq T$ is a right-side boundary decision for data point x_t (and $b_{-1} = 1$). Our goal is to find the best $b_{0:T}$ such that a deep sequence model $q_\phi(\mathbf{x}_T, \mathbf{b}_T)$ optimizes its objective $\mathcal{L}$ under the patching strategy induced by $\mathbf{b}_T$. For example, if $T = 5$ and $\mathbf{b}_T = \{1, 0, 0, 1, 0, 1, 1\}$, then q_ϕ will patch the input sequence according to the following partition: $\mid x_0, x_1, x_2 \mid x_3, x_4 \mid x_5 \mid$ during its embedding stage. Note that the last boundary is relevant in the generative setting, where the objective is $\sum_{t=0}^{T-1} \mathcal{L}(q_\phi(x_{0:t}, b_{-1:t}), x_{t+1})$ and the next point x_{T+1} is generated through $q_\phi(\mathbf{x}_T, \mathbf{b}_T)$ during inference time (see Appendix B for details). The patching policy π_θ that generates boundaries $b_{0:T}$ is then optimized according to the performance of sequence model q_θ w.r.t. $\mathcal{L}$.

For sequence modeling tasks that do not abide by strict time causality during representation learning, such as time series forecasting, masked language modeling, and sequence classifications, the full contextual sequence from initial to terminal state is presented to the model at once. Therefore, the policy network π_θ to decide $b_{0:T}$ can make a joint, collective decision in one step regarding which places to set the patching boundaries, effectively transforming the formulation into a contextual bandit problem. To concretize, we define state $s \triangleq \mathbf{x}_T$ and action $a \triangleq \mathbf{b}_T$, and subsequently formulate the

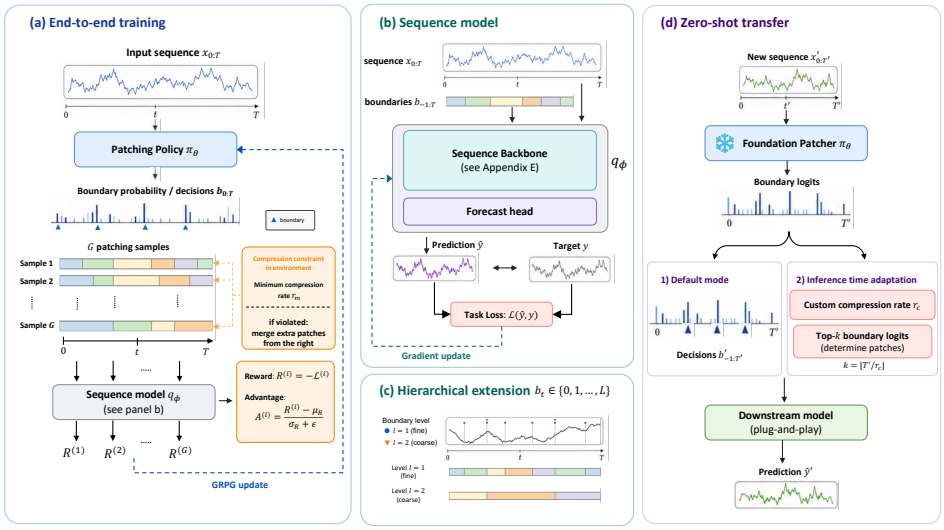


Figure 1: ReinPatch learns a patching **policy** π_θ decoupled from the main sequence **backbone** q_ϕ. **(a)** *Training*: for each input sequence, the policy π_θ emits per-position boundary logits. We sample G boundary sets, each representing a patching strategy of the sequence that the backbone q_ϕ consumes to produce a prediction. The negative task loss on this prediction is then the terminal reward for the corresponding patching strategy. Advantages are computed group-relative and π_θ is updated by GRPG. **(b)** *Sequence model*: encoder downsamples points into patches, a latent transformer models patches, a decoder upsamples patches back to base granularity (Appendix E). A forecast head then generates the prediction to calculate the task loss. Backbone q_ϕ is updated through standard loss back-propagation. **(c)** *Hierarchical*: the same policy can be extended to emit multi-level boundaries in one shot, without having to plan on-the-fly using the hidden representations of q_ϕ throughout its forward pass. **(d)** *Zero-shot transfer*: after large-scale pre-training, learned patching policy π_θ can be frozen and used as a standalone module to produce boundaries for a new downstream backbone, optionally at a custom compression rate via top-k on the boundary logits.

policy and reward as:

$$\pi(a \mid s) \triangleq \pi_\theta\left(b_{0:T} \mid \mathbf{x}_T\right) \cdot \delta_{b_{-1}}, \quad R(s,a) \triangleq -\mathcal{L}\left(q_\theta\left(\mathbf{x}_T, \mathbf{b}_T\right), y\right) \tag{1}$$

where y is the prediction target of the given task, δ_k is the Dirac delta distribution with nonzero density only at k. Under this setup, the reward function depends solely on the sequence model q_ϕ and its canonical objective $\mathcal{L}$, setting the stage for a simple and efficient joint optimization scheme for both π_θ and q_ϕ as we will elaborate in the next section. While our main experiments focus on this non-causal contextual setting, ReinPatch natively supports generative causal modeling via a multi-step MDP formulation. We provide the full unified multi-step MDP formulation in Appendix B.

3.2 Objectives and Optimization

Under this formulation, the simultaneous optimization of q_ϕ and π_θ is clean and straightforward. Since the transition dynamics P and reward function R are only dependent on the main backbone sequence model q_θ and its objective $\mathcal{L}$, the forward pass of q_θ can essentially be seen as the gym environment for π_θ. Hence, after each standard batch gradient update of the sequence model q_θ, its losses can be directly used to compute π_θ's objective. Notice that, similar to Shao et al. [24], we frame the full sequence of actions as a whole in optimization for efficiency, such that they are judged by the same terminal reward and same sequence-level critic, despite having a general MDP formulation (Appendix B). This is essentially what makes group-relative approaches viable and desirable. We use a simple group relative policy gradient algorithm as the objective of π_θ:

$$\nabla_\theta \mathcal{J}_{GRPG} = \frac{1}{G} \sum_{i=1}^{G} \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a^{(i)} \mid s^{(i)}) \frac{R(s_T^{(i)}, a_T^{(i)}) - \mu_R}{\sigma_R + \epsilon}\right] \tag{2}$$

where μ_R and σ_R are the mean and standard deviation of G terminal rewards $\{R(s_T^{(i)}, a_T^{(i)})\}_{1 \le i \le G}$, since training an additional critic network as baseline [15] would often be an overkill for terminal-

reward-only settings, let along one-step MDP settings. For reasons why GRPO [24] is not preferred over GRPG under our setup, see Appendix D.

To put it in words, we sample G different sets of boundaries from policy π_θ for each input sequence, then decide their relative advantage based on how well the sequence model q_ϕ performs on this sequence under each set. Note that standardization can be replaced by centering (i.e. replace $\sigma_R + \epsilon$ with 1) depending on the data characteristics, as the latter can preserve the importance of decisions. By solving patching boundary optimization with RL, we do not rely on soft-discretization to enable gradient flow like prior work such as H-Net [10], which entails that the policy exploration space is continuous, only residing in the local neighborhood of the current boundary policy.

Enforcing desired compression rate To prevent specification gaming, our approach sets a strict minimum compression rate r_m in the *environment* rather than using a soft auxiliary loss as in Hwang et al. [10]. If π_θ proposes boundaries yielding a lower compression rate than r_m, extra patches on the right side will be merged into one during downsampling. See Algorithm 1 in Appendix C for details. This strict enforcement forces the policy to adapt to the constraint in the environment naturally during training, and more importantly, frees practitioners from pre-planning model sizes for the worst-case (uncompressed) scenario and allowing the model to reap the full scaling benefits of patching. We elaborate on the systematic advantages of this strict enforcement in Appendix D.

3.3 Backbone Architecture

For patching policy π_θ, we design it as a standalone lightweight transformer such that it can be easily extracted as a foundation patching policy under the joint pre-training setup, to use for downstream external chunking on smaller datasets. It is composed of an embedding model $m_\theta : \mathbb{R}^d \to \mathbb{R}^{d_{patch}}$ and a final linear layer $f_\theta : \mathbb{R}^{d_{patch}} \to \mathbb{R}^2$ to project the embedding into the corresponding logits for decisions $\{0, 1\}$. In contrast, prior end-to-end patching model such as H-Net [10] has far more complicated patching mechanisms, which rely on the human perception that it is best to set boundaries when context shifts, just like prior external-rule-based and external-model-based methods. This may not necessarily be true for the optimal patching strategy learned by a performance-driven model.

For downstream sequence model q_ϕ, we use a clean and unified architecture for the ease of benchmarking and ablation, demonstrated in Figure 5 of Appendix E. The architecture allows downsampling and upsampling to be done either by cross-attention encoder and decoder, as in Pagnoni et al. [19], or simply scatter and gather plus residual, as in Hwang et al. [10].

Extension to multi-level hierarchical model. The extension to hierarchical model (i.e. more than one level of patching) is elegant under our proposed method, since the patching policy net need not necessarily observe the evolving hidden representation under its downstream sequence model q_ϕ, and therefore is able to plan multi-level patching strategy at once. For sequence model q_ϕ, the extension to hierarchical is trivial (see Appendix E); for patching policy π_θ, we apply a simple modification to final linear layer f_θ such that $f_\theta : \mathbb{R}^{d_{patch}} \to \mathbb{R}^L$ outputs the corresponding logits for extended decision set $b_t \in \{0, 1, \ldots, L\}$ if L levels of patching is desired, where $b_t = l$ means the right end of x_t is the boundary for the first l levels of patching. This is well-defined since boundaries on the j-th level will always be a subset of boundaries on the i-th level, for any $i < j$.

3.4 Inference Time Adaptation

In order to give the foundation patcher additional flexibility to scale across various downstream tasks, where the downstream sequence model width and dataset size could be significantly different from the ones in its pre-training, it should be able to adapt its patching aggressiveness according to a custom compression rate that the practitioners deem more suited to their downstream settings. Hence, in inference mode, we provide the learned patching policy π_θ with the ability to take a target compression rate r_c and adapt accordingly. For contextual tasks, this is relatively simple: we compute the boundary logits $\{l_t\}_{0 \le t \le T}$ with π_θ and pick the indices corresponding to the top-k logits where $k = \lfloor T/r_c \rfloor$. If a target compression rate is not specified by the practitioner, the patching policy calculates the empirical expectation of $\hat{k} = \sum_{t=0}^{T} \left(1 + e^{-l_t}\right)^{-1}$ and pick indices by the top-$\lfloor \hat{k} \rfloor$ logits instead. For generative tasks (i.e. causal tasks), this is more challenging because it is an *online thresholding problem* where we need to hit a global budget (the target compression rate) without seeing the future distribution of the boundary probabilities. The classic approach is to keep a sliding

window of logits (or exponential moving mean and variance) and place patching boundary only if the current logit is higher than the $(1 - 1/r_c)$-th quantile. Note that the yielded compression rate in this case does not strictly adhere to the target compression rate as it does in the contextual tasks, and there are pathological cases that could break this approach (for example, a monotone increasing set of logits, which results in boundary placement everywhere). To this end, we leave further thoughts and development regarding inference time adaptation for generative tasks to future research.

4 Experiments

4.1 Experimental Details

Task. We evaluate ReinPatch on multivariate time series forecasting tasks (problem formulation in Appendix F.2), where the appropriate patching strategy depends both on dataset-level characteristics and on local temporal dynamics. This makes long-term forecasting a suitable setting for comparing different patching strategies. Following prior work [32, 28], we evaluate on the four ETT subsets together with Weather, Electricity, Solar, and Traffic. Details of the datasets used in our forecasting evaluation are presented in Table 6 of Appendix F.

Backbone. To evaluate whether ReinPatch improves forecasting performance relative to existing fixed-size and variable-size patching strategies, we adopt a generic model architecture that has been widely used in patch-based forecasting models as described in Appendix E. Across all methods, we use this model architecture and vary only the patching algorithm as long as its associated encoder-decoder configuration. This controlled protocol isolates the effect of the patching strategy and enables a fair comparison. Additional architecture and selection details are provided in Appendix F.

Baselines. We benchmark ReinPatch against an extensive collection of fixed- and variable-size patching baselines. On the fixed-size side, static patching uses a predetermined uniform patch size [18], selective patching constructs additional features from overlapping fixed-size patches using a scorer [29], and periodic patching chooses the patch size based on signal periodicity in the Fourier domain [26]. On the variable-size side, entropy patching relies on entropy estimation from an external pretrained model [19, 1], TimeSqueeze compression uses local variance in the raw input space [2], and H-Net chunking learns patch boundaries end to end from cosine similarity between Mamba embeddings computed at full resolution [10].

4.2 Main Results

Following standard practice, we report Mean Squared Error (MSE) and Mean Absolute Error (MAE) across four forecasting horizons, $H \in \{96, 192, 336, 720\}$. Because the look-back window size heavily influences performance, we evaluate all methods across look-back lengths $T \in \{96, 192, 336\}$ and report the best validation-selected configuration for each. Additional training details can be found in Appendix F.3.

End-to-end patching. We first evaluate ReinPatch in an end-to-end setting, jointly learning the patching policy and the forecasting model on each dataset. Table 2 presents the best average MSE and MAE across all forecasting horizons. ReinPatch ranks first or second in average MSE on six of the eight datasets, and it gives the best average MSE on ETTh2, ETTm1, and Traffic. The baseline values are averaged over the same five-seed protocol, with baseline standard deviations omitted from the table for readability. These consistent improvements under the shared forecasting backbone indicate that the performance gains stem from the learned, data-driven patching policy rather than architectural modifications.

Zero-shot patching. A broader objective of our work is to explore the possibility of establishing a foundation time series patcher by pre-training a policy on a large-scale corpus for off-the-shelf, zero-shot application on smaller target datasets. To achieve this, we jointly pre-train the patcher and a main sequence forecasting model on the Unified Time Series Dataset (UTSD) [13]. UTSD aggregates diverse, multi-frequency data from public repositories and real-world operations across seven domains: Energy, Environment, Health, IoT, Nature, Transportation, and Web. For downstream evaluation, we deploy this foundation patcher in a zero-shot configuration by freezing its pre-trained weights and exclusively training the task-specific forecasting backbone.

Table 2: Multivariate forecasting average results with forecasting horizons $H \in \{96, 192, 336, 720\}$ for the datasets. **Red**: the best, Blue: the 2nd best. Full results are presented in Table 8 of Appendix F.5.

Datasets	**ETTh1**		**ETTh2**		**ETTm1**		**ETTm2**		**Weather**		**Electricity**		**Solar**		**Traffic**	
Metrics	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
Static Patching [18]	0.409	0.430	0.335	0.383	0.352	0.385	0.272	0.326	0.230	0.267	0.194	0.298	0.208	0.267	0.431	0.304
Selective Patching [29]	0.408	0.430	0.331	0.383	0.359	0.392	0.268	0.324	0.230	0.269	0.187	0.292	0.221	0.278	0.453	**0.286**
Periodic Patching [26]	0.413	0.433	0.336	0.384	0.356	0.388	0.272	0.326	0.228	0.267	0.191	0.294	0.210	0.266	0.449	0.287
Entropy Patching [19, 1]	0.425	0.438	0.339	0.389	0.358	0.386	0.281	0.334	0.230	0.267	0.169	**0.267**	0.203	**0.261**	0.480	0.338
TimeSqueeze's Compression [2]	0.419	0.433	0.334	0.385	0.362	0.387	0.271	0.327	0.229	0.269	0.175	0.275	0.204	0.269	0.502	0.318
H-Net's Chunking [10]	0.434	0.437	0.339	0.387	0.362	0.384	0.272	0.327	0.229	**0.265**	0.175	0.275	0.209	0.275	0.693	0.419
ReinPatch (end-to-end) [Ours]	0.405	0.428	**0.323**	0.378	**0.347**	**0.382**	0.263	**0.321**	0.229	0.268	0.172	0.273	0.202	0.267	**0.415**	0.288
ReinPatch (zero-shot) [Ours]	**0.397**	**0.424**	0.323	**0.377**	0.347	0.382	**0.262**	0.321	**0.226**	0.265	**0.168**	0.267	**0.199**	0.263	0.416	0.286

As shown in Table 2, this zero-shot configuration generally outperforms the end-to-end training of ReinPatch. It is at least tied with the end-to-end variant on seven of eight datasets in average MSE and on all eight datasets in average MAE, yielding the best mean rank among all methods for both metrics. Its mean rank is 1.125 for both MSE and MAE. This suggests that exposing the patching policy to a massive, highly diverse corpus yields more robust and generalizable patches than training exclusively on a narrower target dataset. By freezing the pre-trained weights, the zero-shot patcher avoids dataset-specific overfitting and effectively regularizes the downstream forecasting backbone. This transferability is also by design: the policy is optimized for the instance-normalized signal rather than domain-specific cues such as absolute scale or timestamp spacing, so the structural patterns it learns to segment generalize across domains and distributions.

Across the average results in Table 2, compared to the entropy-based patching baseline, ReinPatch reduces average MSE by 4.9% and average MAE by 3.3% across datasets, with the largest gains on Traffic (13.5% MSE and 15.4% MAE). The per-horizon results in Table 8 of Appendix F.5 show the same trend at finer granularity. Across the 32 dataset-horizon settings, the zero-shot ReinPatch variant ranks first in 19 MSE entries and 15 MAE entries, while the end-to-end variant adds another 5 MSE and 3 MAE first-place entries. Overall, ReinPatch accounts for 42 of the 64 first-place metric entries in the full table. A Friedman-Nemenyi analysis over all 32 settings confirms that ReinPatch's rank advantage is statistically significant against every non-ReinPatch baseline on both metrics (Appendix F.6).

4.3 Ablation and Analysis

Isolating the patching objective. The central question is whether ReinPatch's gains stem from the clean, discrete RL objective itself or merely from the architecture and pre-training exposure. While the main results in Table 2 controlled for most of this exposure, they still allowed varying the encoder/decoder backbone to respect the original literature. In this section, we hold the entire architecture fixed and vary only how boundaries are learned, comparing ReinPatch (zero-shot frozen-UTSD and end-to-end) against three non-RL dynamic patchers: differentiable relaxations of the *same* policy (a straight-through estimator, STE, and Gumbel-Softmax); H-Net chunking run with the *identical* transformer encoder/decoder rather than its SSM variant; and an external entropy patcher pre-trained and frozen under the same UTSD protocol as our zero-shot patcher (Entropy-UTSD). Table 3 reports mean $\pm$ std over five seeds across the 32 (dataset, horizon) settings. The zero-shot ReinPatch variant alone is best on $24/32$ (MSE) and $20/32$ (MAE) settings, rising to $30/32$ and $25/32$ when either ReinPatch variant counts, while the differentiable relaxations, the identical-backbone H-Net, and the UTSD-pretrained entropy patcher win only isolated cells and never lead in aggregate.

Hyperparameter ablation. It is a well-known phenomenon in time series forecasting that performance can often degrade as the look-back window continues to increase, as models struggle to extract robust trends from the accumulated noise [30]. Therefore, adaptive patching must remain effective as context grows. We compare ReinPatch with entropy patching [19, 1] across $T \in \{96, 192, 336, 720\}$ on $H = 96$. As Figure 2 shows, both methods are competitive on short sequence length T, where entropy provides competitive performance relative to ReinPatch. However, ReinPatch consistently pulls ahead on large T, scaling better as context window increases.

This scaling trend highlights a limitation of boundary policies driven by auxiliary metrics such as entropy: the proxy can work for short, simple contexts, but longer windows introduce more heterogeneous patterns and widen the mismatch between estimated entropy and forecasting utility.

Table 3: Controlled ablation of the *patching objective* on a shared transformer backbone and boundary parameterization. We compare ReinPatch (zero-shot frozen-UTSD and end-to-end) against three non-RL dynamic patchers: differentiable relaxations of the *same* policy (straight-through estimator, STE; and Gumbel-Softmax), H-Net's chunking run with the *identical* transformer encoder/decoder (not the SSM variant), and a non-RL external entropy patcher pre-trained and frozen under the same UTSD protocol as our zero-shot patcher (Entropy-UTSD). Cells are mean ± std over 5 seeds. **Red**: best, Blue: 2nd best, per (row, metric).

Models		ReinPatch (zero-shot) [Ours]		ReinPatch (end-to-end) [Ours]		STE Relaxation		Gumbel-Softmax		H-Net (transformer)		Entropy-UTSD	
Metric	H	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	96	**0.360** ±.008	**0.400** ±.006	0.374 ±.002	0.405 ±.004	0.382 ±.007	0.405 ±.005	0.377 ±.005	0.403 ±.004	0.374 ±.005	0.403 ±.002	0.379 ±.001	0.400 ±.001
	192	**0.394** ±.003	**0.419** ±.004	0.410 ±.011	0.423 ±.005	0.416 ±.012	0.430 ±.006	0.428 ±.008	0.434 ±.006	0.422 ±.006	0.424 ±.005	0.441 ±.004	0.432 ±.001
	336	**0.395** ±.007	**0.423** ±.005	0.397 ±.019	0.424 ±.006	0.419 ±.007	0.441 ±.005	0.418 ±.009	0.440 ±.006	0.413 ±.032	0.437 ±.023	0.436 ±.007	0.432 ±.002
	720	**0.440** ±.019	**0.455** ±.010	0.441 ±.029	0.461 ±.007	0.477 ±.008	0.480 ±.005	0.477 ±.018	0.485 ±.013	0.456 ±.008	0.471 ±.005	0.471 ±.012	0.470 ±.007
ETTh2	96	0.281 ±.006	0.339 ±.004	**0.271** ±.003	**0.338** ±.002	0.287 ±.009	0.345 ±.009	0.282 ±.009	0.345 ±.004	0.290 ±.008	0.347 ±.003	0.285 ±.008	0.352 ±.009
	192	0.334 ±.007	0.380 ±.005	**0.330** ±.030	0.374 ±.019	0.331 ±.009	0.383 ±.009	0.336 ±.010	0.384 ±.010	0.333 ±.018	**0.373** ±.010	0.355 ±.009	0.403 ±.007
	336	**0.314** ±.006	**0.374** ±.004	0.322 ±.009	0.380 ±.010	0.334 ±.011	0.394 ±.010	0.323 ±.003	0.382 ±.003	0.323 ±.008	0.383 ±.008	0.344 ±.008	0.402 ±.007
	720	**0.362** ±.008	**0.415** ±.006	0.367 ±.009	0.418 ±.008	0.398 ±.029	0.444 ±.023	0.385 ±.016	0.437 ±.015	0.384 ±.010	0.424 ±.009	0.429 ±.006	0.459 ±.005
ETTm1	96	0.292 ±.004	0.346 ±.003	**0.291** ±.007	**0.345** ±.007	0.301 ±.006	0.354 ±.007	0.300 ±.006	0.352 ±.005	0.296 ±.008	0.347 ±.004	0.308 ±.006	0.353 ±.002
	192	0.330 ±.004	0.371 ±.002	**0.329** ±.004	0.370 ±.003	0.331 ±.004	0.370 ±.002	0.335 ±.006	0.375 ±.003	0.333 ±.004	**0.370** ±.002	0.339 ±.003	0.375 ±.003
	336	**0.357** ±.002	**0.389** ±.001	0.357 ±.003	0.389 ±.002	0.367 ±.012	0.394 ±.005	0.363 ±.005	0.394 ±.004	0.361 ±.002	0.390 ±.001	0.371 ±.002	0.395 ±.003
	720	**0.409** ±.003	**0.422** ±.002	0.410 ±.006	0.423 ±.005	0.418 ±.009	0.426 ±.006	0.416 ±.003	0.427 ±.006	0.414 ±.002	0.425 ±.003	0.422 ±.005	0.425 ±.002
ETTm2	96	**0.168** ±.002	0.257 ±.003	0.168 ±.003	0.257 ±.002	0.170 ±.003	**0.256** ±.001	0.172 ±.002	0.258 ±.002	0.169 ±.003	0.257 ±.004	0.175 ±.002	0.261 ±.001
	192	**0.225** ±.004	**0.297** ±.003	0.228 ±.002	0.299 ±.001	0.232 ±.006	0.299 ±.002	0.231 ±.005	0.299 ±.004	0.229 ±.004	0.300 ±.002	0.243 ±.002	0.310 ±.002
	336	**0.281** ±.007	0.335 ±.006	0.283 ±.012	0.335 ±.007	0.287 ±.007	**0.334** ±.005	0.289 ±.006	0.338 ±.005	0.297 ±.015	0.343 ±.009	0.299 ±.006	0.345 ±.004
	720	0.375 ±.007	0.394 ±.005	**0.374** ±.009	**0.391** ±.005	0.379 ±.002	0.394 ±.004	0.382 ±.011	0.394 ±.006	0.381 ±.007	0.405 ±.006	0.418 ±.008	0.420 ±.005
Weather	96	**0.149** ±.002	**0.203** ±.003	0.153 ±.004	0.206 ±.004	0.155 ±.005	0.208 ±.006	0.157 ±.003	0.210 ±.002	0.155 ±.003	0.207 ±.003	0.155 ±.001	0.208 ±.001
	192	**0.191** ±.001	**0.242** ±.002	0.197 ±.002	0.246 ±.001	0.199 ±.005	0.248 ±.005	0.201 ±.004	0.246 ±.003	0.195 ±.002	0.246 ±.003	0.198 ±.001	0.247 ±.002
	336	0.245 ±.002	**0.282** ±.003	0.244 ±.002	0.282 ±.003	0.245 ±.003	0.283 ±.002	**0.244** ±.001	0.283 ±.003	0.246 ±.002	0.283 ±.004	0.245 ±.002	0.284 ±.002
	720	**0.319** ±.003	**0.334** ±.003	0.321 ±.002	0.336 ±.001	0.320 ±.003	0.335 ±.004	0.322 ±.004	0.336 ±.003	0.322 ±.005	0.337 ±.003	0.322 ±.004	0.337 ±.002
Elec.	96	**0.137** ±.003	0.238 ±.005	0.140 ±.002	0.242 ±.003	0.138 ±.002	0.239 ±.003	0.139 ±.001	0.238 ±.002	0.139 ±.001	0.240 ±.003	0.138 ±.001	**0.238** ±.002
	192	**0.153** ±.001	**0.252** ±.003	0.157 ±.003	0.258 ±.004	0.159 ±.004	0.260 ±.003	0.159 ±.004	0.260 ±.003	0.155 ±.002	0.255 ±.004	0.157 ±.000	0.257 ±.001
	336	0.172 ±.002	0.273 ±.003	0.173 ±.001	0.282 ±.001	0.174 ±.003	0.275 ±.004	0.176 ±.003	0.275 ±.002	**0.172** ±.002	**0.271** ±.004	0.176 ±.000	0.275 ±.002
	720	**0.209** ±.005	**0.304** ±.005	0.217 ±.003	0.312 ±.004	0.223 ±.005	0.322 ±.005	0.227 ±.002	0.324 ±.002	0.221 ±.003	0.316 ±.002	0.214 ±.002	0.306 ±.001
Solar	96	**0.177** ±.003	**0.241** ±.007	0.185 ±.003	0.250 ±.007	0.187 ±.005	0.257 ±.009	0.187 ±.002	0.261 ±.006	0.185 ±.006	0.252 ±.012	0.184 ±.006	0.242 ±.009
	192	**0.199** ±.002	**0.265** ±.002	0.202 ±.003	0.271 ±.005	0.213 ±.005	0.275 ±.008	0.207 ±.003	0.270 ±.002	0.202 ±.003	0.267 ±.009	0.208 ±.009	0.266 ±.005
	336	**0.210** ±.001	0.272 ±.001	0.210 ±.005	0.276 ±.002	0.220 ±.006	0.285 ±.005	0.217 ±.007	0.278 ±.008	0.216 ±.008	0.276 ±.006	0.214 ±.003	**0.269** ±.002
	720	**0.210** ±.000	0.276 ±.002	0.210 ±.002	**0.273** ±.004	0.228 ±.009	0.293 ±.009	0.226 ±.007	0.302 ±.002	0.224 ±.005	0.292 ±.005	0.217 ±.006	0.276 ±.006
Traffic	96	**0.384** ±.002	**0.269** ±.003	0.386 ±.002	0.276 ±.002	0.388 ±.013	0.278 ±.011	0.394 ±.007	0.283 ±.005	0.389 ±.007	0.277 ±.006	0.403 ±.002	0.272 ±.003
	192	**0.403** ±.003	**0.281** ±.001	0.406 ±.005	0.283 ±.004	0.422 ±.009	0.291 ±.006	0.406 ±.006	0.281 ±.010	0.415 ±.003	0.291 ±.003	0.435 ±.001	0.306 ±.007
	336	**0.419** ±.008	**0.286** ±.008	0.420 ±.002	0.289 ±.003	0.435 ±.006	0.305 ±.006	0.438 ±.010	0.299 ±.006	0.429 ±.005	0.303 ±.007	0.448 ±.002	0.316 ±.002
	720	0.459 ±.003	0.309 ±.003	**0.449** ±.002	**0.305** ±.002	0.488 ±.010	0.331 ±.008	0.517 ±.005	0.334 ±.007	0.473 ±.009	0.325 ±.002	0.559 ±.012	0.315 ±.005
1st-place count		**24**	**20**	6	5	0	2	1	0	1	3	0	2

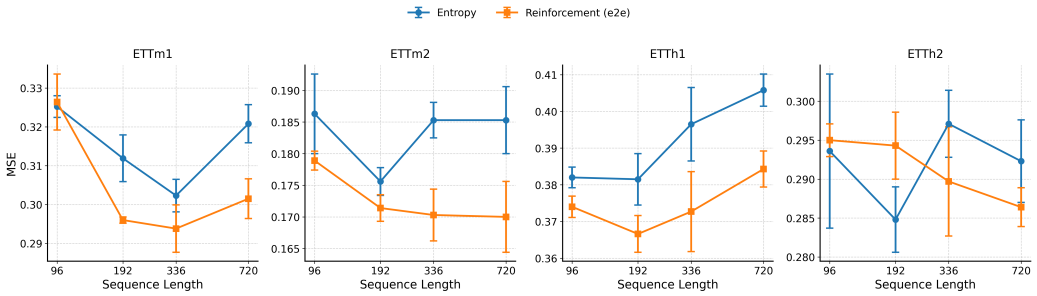


Figure 2: MSE at forecasting horizon $H = 96$ across look-back windows. Entropy patching is competitive for short contexts but struggles as the input context expands, whereas ReinPatch maintains strong performance across most datasets and context lengths.

In contrast, ReinPatch learns boundaries end-to-end for the forecasting objective. The widening gap therefore supports trainable patching as a way to compress long contexts while preserving the temporal detail most useful for prediction. Other than look-back window, our method is not particularly sensitive to hyperparameters, see Figure 3 for details.

Efficiency analysis. Table 4 reports inference-only memory and throughput on ETTh1 with $L = 336$, $H = 720$, and batch size 32. As expected, static patching remains the cheapest baseline because it does not perform input-dependent boundary selection. Nevertheless, ReinPatch incurs only a moderate overhead over static patching, increasing memory from 111.9 to 138.6 MB/sample and reducing throughput from 949.8 to 847.0 samples/s, while retaining the adaptive segmentation behavior that improves forecasting accuracy. Compared with selective patching, ReinPatch achieves higher throughput despite using slightly more memory. More importantly, ReinPatch is substantially more memory-efficient than entropy patching, reducing memory by more than half (138.6 vs. 323.4 MB/sample) while maintaining comparable throughput. These results indicate that the learned patching policy provides a practical efficiency–accuracy trade-off: it preserves the benefits of adaptive

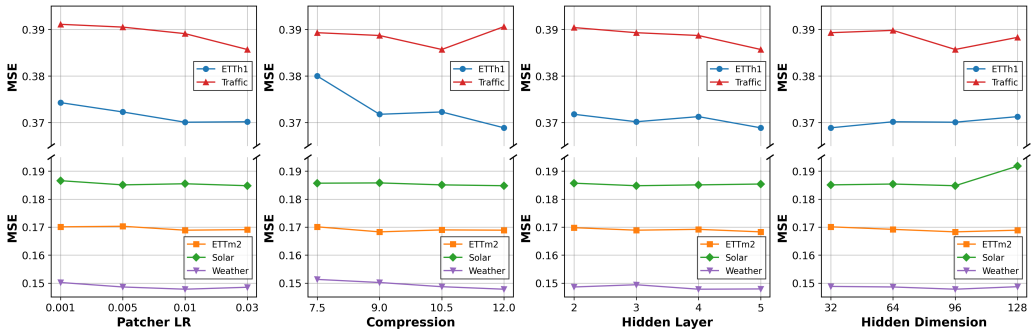


Figure 3: Hyperparameter sensitivity analysis at forecasting horizon $H = 96$; lower MSE is better. From left to right, we vary the patcher learning rate, the minimum compression rate r_m, and the backbone's number of hidden layers and hidden dimension.

tokenization without introducing the large inference footprint of auxiliary-model-based patching. A full breakdown of trainable, total, and patcher parameters together with MACs per method is provided in Table 9 of Appendix F.7.

Table 4: ETTh1 inference efficiency ($L = 336$, $H = 720$).

Method	Memory (MB/sample)	Throughput (samples/s)
Static [18]	111.9	949.8
Selective [29]	113.3	697.6
ReinPatch [Ours]	138.6	847.0
Entropy [1]	323.4	883.6

Table 5: Pairwise permutation-test p-values for fixed-rate boundary-location distributions with target compression rate $r_c = 8$.

	ETTh1	ETTh2	ETTm1	ETTm2	Elec.	Weath.	Solar	Traffic
ETTh1	–	< 0.001	**0.019**	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
ETTh2		–	**0.043**	**0.265**	< 0.001	< 0.001	< 0.001	< 0.001
ETTm1			–	0.003	< 0.001	< 0.001	< 0.001	< 0.001
ETTm2				–	< 0.001	< 0.001	< 0.001	< 0.001
Elec.					–	< 0.001	< 0.001	< 0.001
Weath.						–	< 0.001	< 0.001
Solar							–	< 0.001
Traffic								–

Statistical evidence of data adaptivity. The foundation patcher achieves an average compression rate of 10.92 on UTSD, comfortably above the minimum of 8, indicating that it has internalized the compression constraint and allocates boundaries selectively through pre-training. However, we want to further quantify whether the learned foundation patcher changes *where* it places boundaries across datasets with different characteristics, to investigate whether the patcher has learned specific patterns or is only setting semi-random boundaries to satisfy compression rate. We run the patcher in fixed-rate inference mode with target compression rate $r_c = 8$, which fixes the boundary budget and removes patch count as the main explanation for dataset differences. For each dataset, we estimate a boundary-frequency curve from sampled windows after excluding the forced initial boundary, and compare datasets by the squared distance between their empirical curves. Pairwise significance is assessed with 1000 label permutations within each dataset pair while preserving sample counts; at this resolution, $p = 9.99\times10^{-4}$ means that no permutation produced a difference at least as large as the observed one.

Table 5 summarizes the resulting fixed-rate boundary-location tests across eight datasets. Overall, 27 of 28 dataset pairs are significant at $\alpha = 0.05$. Most pairs reach the minimum attainable Monte Carlo p-value, while the closest significant ETT pair is ETTh2–ETTm1 ($p = 0.043$). The only nonsignificant pair is ETTh2–ETTm2 ($p = 0.265$), indicating similar boundary-location distributions for these two datasets under the fixed-rate control. Because the fixed-rate setting controls the boundary budget, the broad pattern of significant differences shows that ReinPatch adapts boundary locations to dataset structure rather than resorting to trivial patterns that satisfy compression rate.

Interpreting the patching policy. In Figure 4, we visualize the segmentations learned by ReinPatch under the foundation patcher setting on Electricity, Traffic, and Weather. Consistent with some previous heuristics [19, 10, 2], boundaries mainly concentrate near major turning points; however, ReinPatch also *wraps* minor shifts within local neighborhood that preserve seasonality to some extent (see electricity and traffic for example), likely to preserve informative transition shapes and avoid reducing patches to less informative monotone fragments. Furthermore, the policy appears to account for compression loss over long windows by keeping patches structurally compact. It rarely spans

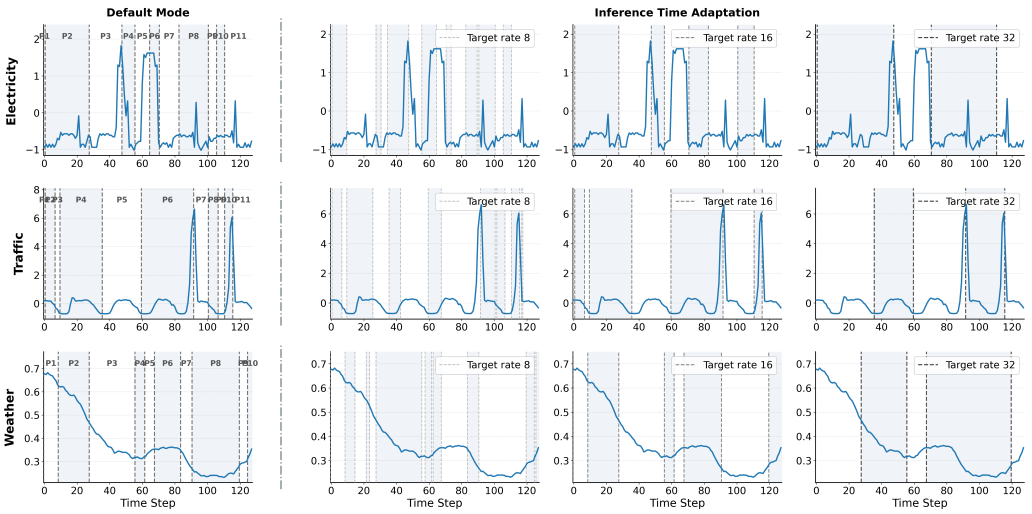


Figure 4: Representative patch segmentations sampled from the foundation patcher trained with a minimum compression rate of 8 on UTSD. The learned boundaries naturally concentrate around major transitions while keeping short neighborhoods of local temporal structure together.

more than a few major turning points, and still places boundaries during a single trend if the same trend persists for an extended period (see the smoother curve on the weather dataset for example).

These patterns help explain why ReinPatch consistently outperforms H-Net, our other end-to-end patching baseline. H-Net was designed for byte-level sequence modeling and places boundaries using cosine similarity between high-dimensional Mamba embeddings at full resolution. For continuous time series, however, useful boundaries should compress the raw trajectory while preserving local transitions and salient events; embedding-space similarity need not align with these raw-signal structures. Thus, H-Net's geometry-driven inductive bias can produce coherent chunks for discrete sequences while missing local structures that matter for forecasting.

5 Conclusion

We introduced Reinforcement Patching (ReinPatch), a novel framework that treats data-adaptive sequence representation as a discrete optimization problem. By utilizing a contextual bandit formulation and Group Relative Policy Gradient (GRPG), ReinPatch bypasses soft-discretization relaxations and cleanly decouples the patching policy from the downstream sequence backbone. Experiments in multivariate time series forecasting demonstrate that ReinPatch consistently outperforms state-of-the-art patching strategies in both joint end-to-end training and zero-shot transfer settings. It scales robustly to extended context windows and provides the community with empirical insights into naturally learned, performance-driven segmentation patterns.

Limitations. First, despite its impressive transferability, our foundation patcher was pre-trained on univariate dynamics. Large-scale, multivariate pre-training to capture cross-channel patching dependencies remains an open challenge within the broader time series modeling umbrella. Second, while the unified one-step MDP formulation enables highly efficient parallelized training, letting the patcher further contextualize its decisions made on-the-fly in the multi-step, causal MDP formulation could still be desirable. Efficiently training deep models in this formulation remains an open challenge.

Future Work. Because the fundamental principles of ReinPatch are domain-agnostic, we plan to explore its potential application as a dynamic, tokenization-free alternative to BPE in large language models, as well as in audio and genomics. This includes adapting RL objectives (e.g., PPO) for settings where the downstream backbone is a massive-scale LLM, and exploring task-specific fine-tuning of pre-trained patching policies. Besides, since the RL formulation is reward-agnostic, the forecasting reward can likewise be replaced with a self-supervised objective (e.g., masked-patch reconstruction), turning the patcher into a self-supervised segmentation policy and a dynamic front-end that feeds variable-length patches to an SSL-pretrained backbone.

Acknowledgments and Disclosure of Funding

We thank Doron Bergman, Rizal Fathony, Nihal Sharma, and Thomas Caputo for the valuable discussions.

References

[1] Sachith Abeywickrama, Emadeldeen Eldele, Min Wu, Xiaoli Li, and Chau Yuen. Entropy Guided Dynamic Patch Segmentation for Time Series Transformers. *arXiv preprint arXiv:2509.26157*, 2025.

[2] 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. In *Recent Advances in Time Series Foundation Models Have We Reached the'BERT Moment'?*, 2025.

[3] 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.

[4] Sam Dauncey and Roger Wattenhofer. You can learn tokenization end-to-end with reinforcement learning. *arXiv preprint arXiv:2602.13940*, 2026.

[5] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. PMLR, 2021.

[7] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[8] 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.

[9] Qihe Huang, Lei Shen, Ruixin Zhang, Jiahuan Cheng, Shouhong Ding, Zhengyang Zhou, and Yang Wang. Hdmixer: Hierarchical dependency with extendable patch for multivariate time series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 12608–12616, 2024.

[10] Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. *arXiv preprint arXiv:2507.07955*, 2025.

[11] Dwipam Katariya, Thomas Caputo, Akshat Shreemali, Juan Manuel Origgi, Nikita Seleznev, Pranab Mohanty, Kalanand Mishra, Nam Nguyen, and James Montgomery. Dp-rec: Towards dynamic patching for efficient long-sequence recommendation. In *Proceedings of the 20th ACM Conference on Recommender Systems*, pp. 182–191, 2026.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[13] Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative Pre-trained Transformers Are Large Time Series Models. In *Forty-first International Conference on Machine Learning*, 2024.

[14] Yong Liu, Guo Qin, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer-XL: Long-Context Transformers for Unified Time Series Forecasting. In *The Thirteenth International Conference on Learning Representations*, 2025.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PmLR, 2016.

[16] Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, 2023.

[17] Peter Bjorn Nemenyi. *Distribution-free multiple comparisons.* Princeton University, 1963.

[18] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations*, 2023.

[19] Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, et al. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, 2025.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[23] 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.

[24] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[25] Kevin Slagle. Spacebyte: Towards deleting tokenization from large language modeling. *Advances in Neural Information Processing Systems*, 37:124925–124950, 2024.

[26] Yihang Wang, Yuying Qiu, Peng Chen, Yang Shu, Zhongwen Rao, Lujia Pan, Bin Yang, and Chenjuan Guo. LightGTS: A Lightweight General Time Series Forecasting Model. In *International Conference on Machine Learning*, pp. 64109–64126. PMLR, 2025.

[27] 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.

[28] 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.

[29] Xingjian Wu, Xiangfei Qiu, Hanyin Cheng, Zhengyu Li, Jilin Hu, Chenjuan Guo, and Bin Yang. Enhancing Time Series Forecasting through Selective Representation Spaces: A Patch Perspective. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[30] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 11121–11128, 2023.

[31] Weijia Zhang, Chenlong Yin, Hao Liu, Xiaofang Zhou, and Hui Xiong. Irregular multivariate time series forecasting: A transformable patching graph neural networks approach. In *Forty-first International Conference on Machine Learning*, 2024.

[32] 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.

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

Appendix

A Extended Related Work

Static patching. Introduced in PatchTST [18], patch-based compression has become a central technique for scaling transformer models for time series. This design has since been adopted by several time-series foundation models, including TimesFM [3], MOMENT [8], Moirai [27], and Timer-XL [14], collectively demonstrating that patching substantially improves training and inference efficiency. However, these approaches rely on a static patch size for an entire input sequence, which limits their ability to model real-world series with substantial temporal heterogeneity.

Data-adaptive patching. Several recent methods have explored adapting patch size automatically rather than selecting it through hand-crafted heuristics. HDMixer [9] enlarges the receptive field by selectively combining adjacent fixed-size patches. IMTS [31] is designed for irregularly sampled series and adjusts patch size to maintain a consistent number of observations per patch. LightGTS [26] selects patch sizes for each input based on periodicity in the Fourier domain, while SRSNet [29] dynamically selects and combines features from several fixed-size patches to improve forecasting performance. Although these approaches introduce adaptivity, they do not explicitly learn within-sequence patching policies that vary patch sizes inside a single series according to local signal characteristics. Furthermore, SRSNet still includes a static patching component while using adaptive patches as auxiliary signal.

Patching using external rules or models. In language modeling, there has been growing interest in moving beyond static offline tokenization schemes such as BPE. Since conventional tokenization can introduce systematic biases and brittle dependencies, several recent models instead operate directly on bytes. Prior work has proposed adaptive segmentation methods that rely on external identifiers or auxiliary models to determine patch boundaries: SpaceByte [25] uses delimiter-like markers, such as spaces, to group bytes into embeddings, while Byte Latent Transformer (BLT) [19] merges predictable byte spans into latent tokens using entropy-guided segmentation. Similar ideas have recently been explored for time series. EntroPE [1] adapts BLT to time series and uses a separately trained entropy model to place boundaries at high-uncertainty time steps, and TimeSqueeze [2] relies on local signal statistics to adjust patch size based on abrupt changes and local variance. However, these approaches depend on modality-specific heuristics, identifiers, or auxiliary models, which prevents the patching strategy from being jointly optimized with the main backbone and may limit its generalization across domains.

End-to-end patching. Beyond approaches that rely on external identifiers, handcrafted statistics, or auxiliary models, a more direct alternative is to learn the patching strategy jointly with the downstream objective. In this line of work, Nawrot et al. [16] introduced a trainable boundary predictor that dynamically pools tokens, but the method suffers from training instabilities that have thus far limited its scalability to larger models and deeper hierarchical settings. More recently, H-Net [10], inspired by U-Net [21], proposed a hierarchical compression architecture that processes sequences at multiple resolutions and uses a state-space model to improve the efficiency of byte-level modeling. Its smoothed routing mechanism enables gradient-based end-to-end patch learning, but because boundary decisions depend on the hidden representation manifold and its evolving dynamics, the patching strategy remains tightly coupled to end-to-end training. This coupling introduces several limitations. First, it prevents a train-short & continuous-train-long regime, which is undesirable when one would like to pre-train a patching module once and freeze its weights during continued pre-training or fine-tuning of the main model. Second, it requires at least one gradient-carrying backbone block at the lowest resolution within the main model. In settings with extremely fine-grained base units, such as character- or byte-level language modeling, this can remain a significant computational bottleneck. H-Net mitigates this issue by using a state-space model at the lowest resolution, but this choice also narrows its applicability, making the approach suited to causal modeling but not non-causal sequence modeling and masked prediction objectives. Independent and concurrent to our work, Dauncey & Wattenhofer [4] investigated the use of reinforcement learning for tokenization in language applications. Aside from different application fields, it differs from ours in optimization algorithm, compression rate enforcement, and hierarchical modeling support. Both H-Net and Dauncey & Wattenhofer [4] have a soft compression rate enforcement, and we argue in Section 3 and Appendix D that it hinders the training framework from fully reaping the benefits of sequence patching models.

B Generative Sequence Patching and Unified MDP Formulation

Recall that the objective is $\sum_{t=0}^{T-1} \mathcal{L}(q_\phi(x_{0:t}, b_{-1:t}), x_{t+1})$ in generative modeling, and the next point x_{T+1} is generated through $q_\phi(\mathbf{x}_T, \mathbf{b}_T)$ during inference time. Since patching decisions are sequential under this setting, the natural formulation is a sequential decision process as follows.

Generative sequence patching as multi-step MDP Define state and action at step t as

$$s_t \triangleq \left(x_{0:t}, b_{-1:(t-1)}\right), \quad a_t \triangleq b_t, \tag{3}$$

The MDP is characterized by the following patching policy, transition dynamics, and reward function:

$$\pi(a_t \mid s_t) \triangleq \pi_\theta\left(b_t \mid x_{0:t}, b_{-1:(t-1)}\right) \tag{4}$$

$$P(s_{t+1} \mid s_t, a_t) \triangleq p(x_{t+1} \mid x_{0:t}) \cdot \delta_{x_{0:t}} \cdot \delta_{b_{-1:t}} \tag{5}$$

$$R(s_t, a_t) \triangleq -\mathcal{L}\left(q_\phi\left(x_{0:t}, b_{-1:t}\right), x_{t+1}\right). \tag{6}$$

In training, the sequence transition dynamic $p(x_{t+1} \mid x_{0:t})$ is induced by the data; in inference, sequential generation is performed by approximating $p(x_{t+1} \mid x_{0:t})$ with the learned model $q_\phi\left(x_{0:t}, b_{-1:t}\right)$.

Although conditioning decision b_t on the previously sampled boundary decisions $b_{0:(t-1)}$ could be more expressive, such a strategy is significantly less efficient to train for long context windows, especially with a transformer backbone. Therefore, similar to H-Net [10], we let boundary decisions at steps t_1 and t_2 be independently sampled for $t_1 \neq t_2$, yielding the simplified factorization below.

Unified one-step MDP for efficient optimization Let $s_t \triangleq x_{0:t}$ in place of the definition in Eq. 3. The MDP is characterized by

$$\pi(a_t \mid s_t) \triangleq \pi_\theta\left(b_t \mid x_{0:t}\right) \tag{7}$$

$$P(s_{t+1} \mid s_t, a_t) \triangleq p(x_{t+1} \mid x_{0:t}) \cdot \delta_{x_{0:t}} \tag{8}$$

$$R(s_t, a_t) \triangleq \begin{cases} -\sum_{i=0}^{t} \mathcal{L}\left(q_\phi\left(x_{0:i}, b_{-1:i}\right), x_{i+1}\right), & \text{if } t = T \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Although the formulation itself still appears to be a multi-step MDP, this modification enables parallelized decision making and reward computation during training and makes the MDP a realization of the contextual bandit in Eq. 1, since the trajectory

$$\tau(\theta) = P(s_0) \prod_{t=0}^{T} \pi(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t) = p(\mathbf{x}_T) \prod_{t=0}^{T} \pi_\theta(b_t \mid x_{0:t}) \tag{10}$$

can be computed in one forward pass of π_θ, and the terminal reward can then be assessed in one forward pass of q_ϕ. We use this unified setup as our formulation, where causality can be enforced if desired so long as the models satisfy

$$q_\phi\left(\mathbf{x}_T, \mathbf{b}_T\right)_{[t]} = q_\phi\left(x_{0:t}, b_{-1:t}\right), \quad \pi_\theta\left(b_{0:t} \mid \mathbf{x}_T\right)_{[t]} = \pi_\theta\left(b_t \mid x_{0:t}\right), \tag{11}$$

where $M_{[t]}$ is the t-th column vector (or element) of matrix (or vector) M. This is automatically the case for standard causal sequence backbones such as uni-directional SSMs and transformers with causal masks. During inference, one can compute the right-side boundary logits of each time step sequentially, as in Eq. 7, on tasks where transition dynamic $p(x_{t+1} \mid x_{0:t})$ needs to be approximated by sequence model $q_\phi\left(x_{0:t}, b_{-1:t}\right)$.

Note that the one-step MDP view is a *training-time reward-assignment scheme*, not a claim that boundary decisions have to be made non-sequentially: the sharing of one terminal reward across actions is what makes the group-relative advantage well-defined. The same formulation reduction applies to GRPO for sequential language-model generation, whose outcome-supervised objective is likewise equivalent, in its reward assignment, to a contextual bandit in which the entire sampled text output can be viewed as a single action receiving one terminal reward. A causal ReinPatch therefore follows by masking the policy to attend only to $x_{\leq t}$, which leaves the reward, advantage, and GRPG update unchanged and emits boundaries in a single pass at inference, similar to H-Net [10].

Algorithm 1 Joint End-to-End Training of ReinPatch and Downstream Backbone

Require: Dataset $\mathcal{D}$, initialized patching policy π_θ, initialized downstream sequence model q_ϕ
Require: Group size G, minimum compression rate r_m, learning rates η_θ, η_ϕ

while not converged **do**
  Sample a mini-batch of sequences $\mathbf{x}_T$ and targets y from $\mathcal{D}$
  for each sequence in the mini-batch **do**
    for $i = 1, \ldots, G$ **do**
      Sample boundary decisions $b_{0:T}^{(i)} \sim \pi_\theta(\cdot \mid \mathbf{x}_T)$ ▷ The policy for q_ϕ
      if compression rate $(T+1)/\sum_{t=0}^{T} b_t^{(i)} < r_m$ **then** ▷ The environment for π_θ
        Merge extra patches from the right side of $\mathbf{b}_T^{(i)}$
      end if
      Compute backbone prediction $\hat{y}^{(i)} = q_\phi(\mathbf{x}_T, \mathbf{b}_T^{(i)})$
      Compute task loss $\mathcal{L}^{(i)} = \mathcal{L}(\hat{y}^{(i)}, y)$
      Set terminal reward $R^{(i)} = -\mathcal{L}^{(i)}$
    end for
    Calculate group reward statistics:
    $\mu_R = \frac{1}{G}\sum_{i=1}^{G} R^{(i)}$ and $\sigma_R = \sqrt{\frac{1}{G}\sum_{i=1}^{G}(R^{(i)} - \mu_R)^2}$
    for $i = 1, \ldots, G$ **do**
      Compute advantage $A^{(i)} = \frac{R^{(i)} - \mu_R}{\sigma_R + \epsilon}$
    end for
    Compute policy gradient (GRPG): ▷ For policy π_θ update
    $g_\theta = \frac{1}{G}\sum_{i=1}^{G} \nabla_\theta \log \pi_\theta(b_{0:T}^{(i)} \mid \mathbf{x}_T) A^{(i)}$
    Compute backbone gradient: ▷ For environment q_ϕ update
    $g_\phi = \nabla_\phi \frac{1}{G}\sum_{i=1}^{G} \mathcal{L}^{(i)}$
  end for
  Average gradients g_θ and g_ϕ across the mini-batch
  Update policy network: $\theta \leftarrow \theta + \eta_\theta g_\theta$
  Update backbone network: $\phi \leftarrow \phi - \eta_\phi g_\phi$
end while

C Training Algorithm

An algorithm block for the training workflow of our approach is presented in Algorithm 1.

D Further Modeling Discussion

Why is GRPO not preferred over GRPG? The design of group mean as critic instead of neural network as critic is most recently well-known under the LLM mathematical reasoning line of work as group relative policy optimization (GRPO) [24]:

$$\mathcal{J}_{GRPO} = \frac{1}{G}\sum_{i=1}^{G} \mathbb{E}\left[\sum_{t=0}^{T} \min\left\{r_\theta A^{(i)}, \text{clip}\left\{r_\theta A^{(i)}, 1-\delta, 1+\delta\right\}\right\}\right] - \beta D_{KL}\left[\pi_\theta \parallel \pi_{ref}\right] \quad (12)$$

where $r_\theta = \frac{\pi_\theta(a_t^{(i)} | s_t^{(i)})}{\pi_{\theta_{old}}(a_t^{(i)} | s_t^{(i)})}$ and $A^{(i)} = \frac{R(s_T^{(i)}, a_T^{(i)}) - \mu_R}{\sigma_R + \epsilon}$. On the other hand, The GRPG objective in Eq. 2 can be written as

$$\nabla_\theta \mathcal{J}_{GRPG} = \frac{1}{G}\sum_{i=1}^{G} \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \frac{R(s_T^{(i)}, a_T^{(i)}) - \mu_R}{\sigma_R + \epsilon}\right] \quad (13)$$

if we let $a_t \triangleq b_t$ and $s_t \triangleq x_{0:T}$ (or $s_t \triangleq x_{0:t}$ in the unified one-step MDP setup for the causal setting described in Appendix B). Despite having the same advantage function, Eq. 12 has two distinct components compared to Eq. 13: 1) the PPO [22] objective in place of policy gradient for multi-epoch mini-batch update, 2) reference policy π_{ref} as anchor.

For the first part, the reason single-epoch updates are safer (hence, policy gradient is preferred over PPO) is that the environment in our case is a continuously evolving and highly parallelizable neural model q_θ. Algorithms like PPO do multi-epoch mini-batch updates to squeeze as much value as possible out of a batch of rollouts because interacting with their environments (e.g. physics simulators or math verifiers) is slow and expensive. In our setup, sample generation is cheap. Doing a single update step on fresh data (i.e. pure on-policy learning) prevents the policy from taking overly large, destructive steps and avoids the need for complex clipping mechanisms.

For the second part, the answer is simple: this work focuses on the from-scratch training and pre-training of sequence patching policy, where we do not have a pre-trained reference policy and we do not explore fine-tuning patching policy under the scope of this paper. We have an online learning setup where the reward function fully capture the intended goal of the patching policy – to find the patching boundaries that result in the best sequence model performance. Besides, contrary to conversational AI such as LLMs, we do not require diversity in its generation. It is not a problem if the optimized patcher always provides the same patching strategy for a given sequence. If early exploration is of particular concern, practitioners are free to apply standard exploration strategies as well to prevent premature convergence, such as adding entropy term $\mathcal{H}(\pi_\theta)$ with coefficient β to the objective function instead of $\beta D_{KL}\left[\pi_\theta \parallel \pi_{ref}\right]$.

How is desired compression rate enforced? In prior work such as Hwang et al. [10], a target compression rate is encouraged by an auxiliary loss $\mathcal{L}_{aux}$ that increases when the compression rate under the sampled patching boundaries deviates from the target compression rate. This prevents model from converging to a trivial solution, such as placing patching boundaries everywhere, or not placing boundaries at all (since the backbone model has residual connection). On the contrary, the objective of our patching strategy is solely to optimize the performance of the downstream sequence model q_θ, so how can such reward hacking be prevented?

To answer this question, we first argue that it is systematically better to set desired compression rate as a strict limit rather than as a soft auxiliary objective as in H-Net. The reason is its advantage in model size pre-planning. Soft-encouragement of compression rate means practitioners always need to pre-plan model dimensions for the worst case scenario, which is having no compression (i.e. model places patching boundaries everywhere) in the beginning of training, or at any point during training. However, one of the key benefits of patching is the scalability: it allows a deeper neural model with extended hidden size as the spatial or temporal horizons concentrate, and H-Net is not able to fully reap such benefit. After all, if the algorithm truly thinks that setting boundaries everywhere is going to yield the best performance, then it should be allowed to do so. The real problem here is that it cannot enlarge model size during the course of training to adapt to the shrinking context length to fully understand the benefit of doing compression.

On the contrary, setting strict minimum compression rate can be achieved under our proposed method, since we do not rely on gradient flow from the main model to update the patcher. We enforce the minimum compression rate r_m in q_ϕ, in a way such that if π_θ proposes a set of boundaries that yields a lower compression rate than r_m, all extra patches on the right side are merged into one patch. In other words, the information regarding the preset compression rate requirement is in the *environment* of π_θ, and π_θ has to figure it out and adapt to it during the course of training. Note that we perform merging on the right side to make the environment more challenging for π_θ, since recent information are usually perceived to be more meaningful for sequence data. With this strict enforcement, we need not pre-plan model sizes for the worst case scenario, and can reap the full benefits of deep patching models.

E Sequence Model Architecture

We use a standard sequence patching backbone model widely used in prior work, illustrated in Figure 5. For details on cross-attention encoding and decoding, see Pagnoni et al. [19]; for details on the scatter and gather operations, see Hwang et al. [10]. This design extends to hierarchical patching when multiple levels of patching boundaries are provided by stacking encoders and decoders on downsampled and upsampled features, analogous to how multi-resolution architectures such as U-Net stack operations across resolution levels. We use a two-level hierarchical model for our end-to-end patching method in the main experiments.

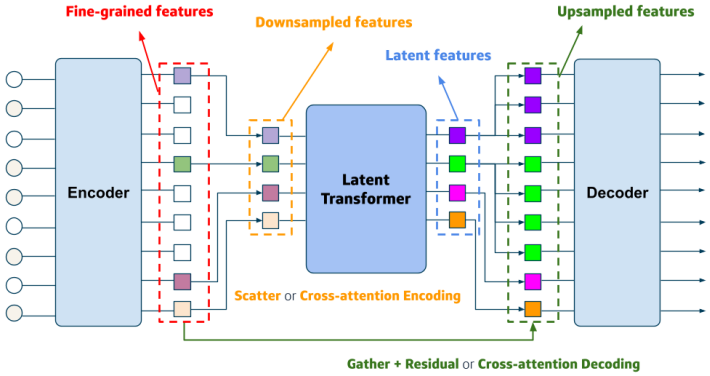


Figure 5: Sequence backbone model. Note that this illustration is for contextual tasks. For generative (i.e. causal) tasks, patches are right-shifted by one (and a special token embedding is added to the beginning of the sequence) before upsampling.

For downsampling, we share the view of Hwang et al. [10] that cross-attention encoding in Pagnoni et al. [19] is not always necessary, but we maintain that cross-attention decoding can be beneficial. The gather upsampling approach, which repeats the same hidden state for every point in a patch, is less compatible with down-projection and can introduce information loss. In U-Net-style architectures, the consecutive operations of up-projection and downsampling, as well as upsampling and down-projection, allow information to transfer between spatial and hidden dimensions to mitigate compression loss. This does not hold for the combination of gather upsampling and down-projection, since information compressed in hidden dimensions cannot transfer back to temporal dimensions. This reduces the benefit of hierarchical modeling and increases reliance on the residual connection.

F Experiment Details

F.1 Datasets

Table 6 summarizes the statistics of the datasets used in our forecasting evaluation, including their domain, sampling frequency, length, dimensionality, and train/validation/test split.

Table 6: Statistics of datasets.

Dataset	Domain	Frequency	Lengths	Dim	Split	Description
ETTh1	Electricity	1 hour	14,400	7	6:2:2	Power transformer 1, comprising seven indicators such as oil temperature and useful load
ETTh2	Electricity	1 hour	14,400	7	6:2:2	Power transformer 2, comprising seven indicators such as oil temperature and useful load
ETTm1	Electricity	15 mins	57,600	7	6:2:2	Power transformer 1, comprising seven indicators such as oil temperature and useful load
ETTm2	Electricity	15 mins	57,600	7	6:2:2	Power transformer 2, comprising seven indicators such as oil temperature and useful load
Weather	Environment	10 mins	52,696	21	7:1:2	Recorded every day for the whole year 2020, which contains 21 meteorological indicators
Electricity	Electricity	1 hour	26,304	321	7:1:2	Electricity records the electricity consumption in kWh every 1 hour from 2012 to 2014
Solar	Energy	10 mins	52,560	137	7:1:2	Solar production records collected from 137 PV plants in Alabama
Traffic	Traffic	1 hour	17,544	862	7:1:2	Road occupancy rates measured by 862 sensors on San Francisco Bay area freeways

F.2 Problem Formulation

Consider a multivariate time series $\mathbf{x}_T = x_{0:T}$ with C channels, yielding an input $\mathbf{x}_T \in \mathbb{R}^{C \times T}$. The forecasting model takes the most recent T time steps from each channel as input and predicts the next H time steps for all channels. Following [18], we adopt the channel-independence principle and process the multivariate input as a collection of univariate series with shared model parameters, then aggregate the per-channel predictions back into a multivariate forecast. This is the standard paradigm established by recent forecasting architectures [18, 30, 29].

Table 7: Architecture components of different forecasting methods.

Method	Encoder ($\times$levels)	Decoder ($\times$levels)	Latent Model
Static Patching	Transformer $\times 1$	Transformer $\times 1$	Transformer
Selective Patching	Transformer $\times 1$	Transformer $\times 1$	Transformer
Periodic Patching	Transformer $\times 1$	Transformer $\times 1$	Transformer
Entropy Patching	Transformer $\times 1$	Transformer $\times 1$	Transformer
TimeSqueeze's Compression	SSM $\times 1$	SSM $\times 1$	Transformer
H-Net's Chunking	SSM $\times 2$	SSM $\times 2$	Transformer
ReinPatch (end-to-end) [Ours]	Transformer $\times 2$	Transformer $\times 2$	Transformer
ReinPatch (zero-shot) [Ours]	Transformer $\times 1$	Transformer $\times 1$	Transformer

F.3 Training and Selection Details

For all forecasting experiments, we use the official train/validation/test splits associated with each benchmark dataset and select hyperparameters by validation MSE. Candidate look-back lengths are $T \in \{96, 192, 336\}$, and all tables report the test result from the validation-selected checkpoint and configuration. Unless otherwise stated, runs use random seeds $\{1, 2, 3, 4, 5\}$; reported means are seed averages and reported $\pm$ values are standard deviations across seeds.

The downstream forecasting backbone is optimized with Adam and MSE loss. Forecasting runs use OneCycleLR, early stopping on validation loss, batch sizes selected from $\{32, 64, 128, 256\}$ depending on the dataset and horizon, and training budgets between 20 and 80 epochs with patience between 7 and 20. All forecasting experiments of ReinPatch are conducted using PyTorch [20] in Python 3.11 and executed on an NVIDIA A100 GPU.

For the zero-shot patching experiments, the foundation patcher is jointly pre-trained with a forecasting backbone on UTSD using sequence length 240, prediction length 96, two epochs, batch size 1024, Adam learning rate 10^{-4} for the backbone, patcher learning rate 3×10^{-3}, group size 3, and minimum compression rate 8. The pre-trained patcher uses four transformer layers, hidden size 32, two attention heads, and dropout 0.1. Pre-training are run across two distributed nodes, each with eight NVIDIA A100 GPUs. During downstream zero-shot evaluation, the patcher weights are frozen and only the task-specific forecasting backbone is trained.

Our code is available in the supplementary material.

F.4 Encoder-Decoder configuration

Based on the baseline chosen, patching is implemented via either a cross-attention-based transformer or an SSM. For TimeSqueeze's compression and H-Net's chunking, we use an SSM backbone for the downsampling encoder and upsampling decoder, aligned with the original literature [2, 10]. For the remaining patching methods, we use a transformer backbone. Exact configurations for each method are listed in Table 7. We choose only one level of compression for the ReinPatch foundation zero-shot patcher to keep it transferable and compatible with forecasting methods that may not support multi-level hierarchical patching. H-Net and ReinPatch are the only evaluated methods capable of supporting multi-level hierarchical patching.

F.5 Full Results

While Table 2 in the main text reports MSE and MAE averaged over the four forecasting horizons, Table 8 presents the complete per-horizon breakdown ($H \in \{96, 192, 336, 720\}$) for all eight methods across the eight datasets, with each cell averaged over five seeds. The finer granularity confirms the trend seen in the averaged results: ReinPatch (zero-shot and end-to-end) takes the majority of first-place entries and stays competitive at both short and long horizons, indicating that its averaged advantage does not stem from a few favorable horizons but holds consistently as the forecasting difficulty grows.

F.6 Aggregate Significance

With five seeds per setting, a per-cell two-sided Wilcoxon test is uninformative (its smallest attainable $p \approx 0.06$), so we assess significance in aggregate using the standard multi-dataset comparison

Table 8: Multivariate forecasting full results with forecasting horizons $H \in \{96, 192, 336, 720\}$ for the datasets. **Red**: the best, Blue: the 2nd best.

Models		ReinPatch (zero-shot) [Ours]		ReinPatch (end-to-end) [Ours]		Static Patching [18]		Selective Patching [29]		Periodic Patching [26]		Entropy Patching [19, 1]		TimeSqueeze's Compression [2]		H-Net's Chunking [10]	
Metrics	F.H.	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	96	**0.360** ± 0.008	**0.400** ± 0.006	0.374 ± 0.007	0.405 ± 0.006	0.364	0.402	0.363	0.401	0.366	0.404	0.378	0.401	0.367	0.400	0.384	0.401
	192	**0.394** ± 0.003	**0.419** ± 0.004	0.410 ± 0.016	0.423 ± 0.009	0.415	0.427	0.417	0.428	0.419	0.431	0.441	0.440	0.429	0.425	0.436	0.432
	336	**0.395** ± 0.007	0.423 ± 0.005	0.397 ± 0.007	0.424 ± 0.006	0.402	0.427	0.404	0.427	0.395	**0.421**	0.424	0.445	0.420	0.439	0.438	0.434
	720	**0.440** ± 0.019	**0.455** ± 0.010	0.441 ± 0.016	0.461 ± 0.011	0.453	0.463	0.447	0.462	0.471	0.478	0.457	0.465	0.459	0.468	0.478	0.479
ETTh2	96	0.281 ± 0.006	0.339 ± 0.004	0.271 ± 0.005	0.338 ± 0.006	**0.265**	**0.328**	0.268	0.334	0.288	0.345	0.283	0.349	0.270	0.338	0.284	0.341
	192	0.334 ± 0.007	0.380 ± 0.005	0.330 ± 0.010	**0.374** ± 0.007	0.335	0.380	0.328	0.376	0.332	0.378	0.335	0.381	**0.328**	0.377	0.337	0.385
	336	**0.314** ± 0.006	**0.374** ± 0.004	0.322 ± 0.009	0.380 ± 0.010	0.331	0.385	0.320	0.378	0.326	0.386	0.329	0.383	0.325	0.383	0.327	0.384
	720	**0.362** ± 0.008	**0.415** ± 0.006	0.367 ± 0.008	0.418 ± 0.007	0.409	0.439	0.405	0.444	0.398	0.428	0.410	0.442	0.412	0.442	0.410	0.440
ETTm1	96	0.292 ± 0.004	0.346 ± 0.003	**0.291** ± 0.005	**0.345** ± 0.003	0.293	0.347	0.302	0.354	0.307	0.355	0.310	0.353	0.311	0.354	0.299	0.347
	192	0.330 ± 0.004	0.371 ± 0.002	**0.329** ± 0.003	**0.370** ± 0.003	0.334	0.376	0.351	0.388	0.335	0.375	0.335	0.375	0.338	0.371	0.343	0.371
	336	**0.357** ± 0.002	**0.389** ± 0.001	0.357 ± 0.003	0.389 ± 0.002	0.371	0.398	0.371	0.401	0.370	0.397	0.369	0.393	0.371	0.392	0.381	0.394
	720	0.409 ± 0.003	0.422 ± 0.002	0.410 ± 0.002	0.423 ± 0.001	**0.409**	**0.420**	0.413	0.423	0.414	0.426	0.418	0.423	0.428	0.433	0.427	0.423
ETTm2	96	**0.168** ± 0.002	**0.257** ± 0.003	0.168 ± 0.002	0.257 ± 0.002	0.174	0.261	0.172	0.261	0.172	0.259	0.176	0.263	0.175	0.265	0.177	0.265
	192	**0.225** ± 0.004	**0.297** ± 0.003	0.228 ± 0.007	0.299 ± 0.004	0.232	0.306	0.227	0.297	0.235	0.302	0.244	0.310	0.232	0.308	0.229	0.300
	336	**0.281** ± 0.007	**0.335** ± 0.006	0.283 ± 0.008	0.335 ± 0.003	0.300	0.349	0.296	0.345	0.286	0.337	0.298	0.346	0.291	0.339	0.299	0.340
	720	0.375 ± 0.007	0.394 ± 0.005	**0.374** ± 0.004	0.391 ± 0.003	0.380	**0.388**	0.379	0.394	0.394	0.404	0.408	0.416	0.386	0.398	0.382	0.404
Weather	96	**0.149** ± 0.002	0.203 ± 0.003	0.153 ± 0.003	0.206 ± 0.002	0.155	0.205	0.155	0.207	0.153	0.206	0.157	0.205	0.151	0.202	0.151	**0.201**
	192	**0.191** ± 0.001	**0.242** ± 0.002	0.197 ± 0.003	0.246 ± 0.004	0.195	0.243	0.197	0.246	0.196	0.244	0.198	0.246	0.197	0.248	0.196	0.242
	336	0.245 ± 0.002	0.282 ± 0.003	**0.244** ± 0.001	0.282 ± 0.001	0.250	0.285	0.246	0.283	0.248	0.286	0.247	0.284	0.244	0.286	0.246	**0.281**
	720	0.319 ± 0.003	0.334 ± 0.003	0.321 ± 0.002	0.336 ± 0.002	0.319	0.335	0.323	0.338	0.316	**0.330**	**0.316**	0.331	0.323	0.341	0.324	0.335
Electricity	96	**0.137** ± 0.003	**0.238** ± 0.005	0.140 ± 0.003	0.242 ± 0.005	0.151	0.264	0.150	0.260	0.152	0.265	0.139	0.240	0.140	0.242	0.141	0.246
	192	**0.153** ± 0.001	**0.252** ± 0.003	0.157 ± 0.003	0.258 ± 0.004	0.182	0.292	0.175	0.282	0.177	0.283	0.157	0.256	0.163	0.264	0.159	0.258
	336	0.172 ± 0.002	0.273 ± 0.003	0.173 ± 0.001	0.282 ± 0.001	0.202	0.307	0.194	0.298	0.198	0.301	**0.172**	**0.270**	0.179	0.277	0.176	0.274
	720	0.209 ± 0.005	0.304 ± 0.005	0.217 ± 0.003	0.312 ± 0.004	0.240	0.331	0.230	0.326	0.236	0.325	**0.208**	**0.301**	0.217	0.318	0.223	0.323
Solar	96	**0.177** ± 0.003	**0.241** ± 0.007	0.185 ± 0.003	0.250 ± 0.007	0.192	0.248	0.181	0.251	0.193	0.249	0.180	0.247	0.187	0.251	0.191	0.265
	192	**0.199** ± 0.002	0.265 ± 0.002	0.202 ± 0.003	0.271 ± 0.005	0.207	0.277	0.231	0.286	0.205	0.271	0.202	**0.251**	0.201	0.273	0.206	0.274
	336	0.210 ± 0.001	0.272 ± 0.001	0.210 ± 0.005	0.276 ± 0.002	0.215	0.270	0.233	0.282	**0.209**	**0.269**	0.221	0.279	0.220	0.275	0.215	0.276
	720	0.210 ± 0.000	0.276 ± 0.002	0.210 ± 0.002	0.273 ± 0.004	0.218	0.272	0.239	0.292	0.231	0.273	0.211	**0.266**	**0.209**	0.276	0.224	0.286
Traffic	96	**0.384** ± 0.002	**0.269** ± 0.003	0.386 ± 0.002	0.276 ± 0.002	0.425	0.323	0.400	0.275	0.405	0.271	0.426	0.311	0.433	0.297	0.573	0.377
	192	**0.403** ± 0.003	**0.281** ± 0.001	0.406 ± 0.005	0.283 ± 0.004	0.407	0.288	0.420	0.282	0.446	0.282	0.435	0.307	0.455	0.299	0.770	0.460
	336	**0.419** ± 0.008	0.286 ± 0.008	0.420 ± 0.002	0.289 ± 0.003	0.419	**0.283**	0.458	0.284	0.444	0.289	0.538	0.392	0.553	0.350	0.866	0.500
	720	0.459 ± 0.003	0.309 ± 0.003	**0.449** ± 0.002	0.305 ± 0.002	0.471	0.321	0.535	**0.302**	0.503	0.308	0.522	0.343	0.566	0.324	0.561	0.337
1^{st} Count		**19**	**15**	5	3	2	4	0	1	1	3	3	4	2	0	0	2

protocol [5]: a Friedman test [7] with a Nemenyi post-hoc [17] over all 32 (dataset, horizon) settings of Table 8. The Friedman test rejects rank-equality on both metrics (MSE $p \approx 4 \times 10^{-15}$; MAE $p \approx 8 \times 10^{-8}$). By the Nemenyi post-hoc (critical difference 1.86 at $\alpha = 0.05$), zero-shot ReinPatch attains the best mean rank among the eight methods (1.73 for MSE, 2.22 for MAE) and is significantly ahead of all six non-ReinPatch baselines on both metrics; the zero-shot and end-to-end variants are statistically indistinguishable from each other. This complements the boundary-adaptivity permutation test of Table 5 ($p \approx 10^{-3}$).

F.7 Compute and Model Complexity

Table 9 breaks down trainable, total, main-model, and patcher parameters together with multiply-accumulate operations (MACs), anchored at ETTh1 ($L = 336 \rightarrow H = 720$) with every method on the unified backbone and differing only in the patcher. ReinPatch's patcher adds only ∼49K parameters on top of the ∼2.25M main-model parameters, and its MACs (5.06G) are close to the static lower bound (4.79G) and well below the external entropy patcher (7.27G), which must additionally run a separate pre-trained entropy model. The differentiable relaxations (STE, Gumbel-Softmax) share ReinPatch's parameter count but incur slightly higher MACs, confirming that ReinPatch's accuracy gains do not come from added model capacity or compute.

Table 9: Compute and model complexity per patching method, anchored at ETTh1 (look-back 336 → horizon 720). Every method runs on the unified backbone under the same protocol, differing only in the patcher. MACs reported in giga-operations.

Method	Trainable Params	Total Params	Main-Model Params	Patcher Params	MACs (G)
ReinPatch (end-to-end) [Ours]	2,296,889	2,296,889	2,247,782	49,107	5.055
STE Relaxation	2,296,889	2,296,889	2,247,782	49,107	5.148
Gumbel-Softmax	2,296,889	2,296,889	2,247,782	49,107	5.148
Static Patching	2,012,638	2,012,638	2,012,622	16	4.791
Entropy Patching	2,014,670	2,050,670	2,012,622	38,048	7.272

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the proposed method, training setting, and empirical scope; Sections 3 and 4 provide the corresponding method and evidence.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 includes a limitations paragraph discussing the current pre-training scope and the remaining challenge of efficient multi-step causal formulations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper proposes a modeling and optimization framework but does not state formal theoretical results requiring proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4 describes the task, datasets, baselines, horizons, look-back lengths, and seed averaging; Appendix F and Table 7 provide additional formulation and architecture details.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The anonymized supplemental package with source code is included along with submission.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4, Table 6, Appendix F, and Table 7 specify the evaluation datasets, splits, horizons, look-back lengths, baselines, and model configurations used for the reported results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 4.2 states that results are averaged over five random seeds, and Table 8 reports $\pm$ variability for the ReinPatch variants across forecasting horizons.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Table 4 reports inference GPU memory and throughput for one efficiency setting, but the manuscript does not provide full training hardware, memory, or runtime information for each experiment.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public time-series forecasting benchmarks, cites prior datasets and methods, and does not involve human-subject data collection or deployment claims.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: The paper presents a foundational sequence patching method evaluated on forecasting benchmarks and does not study deployment in a societal decision-making setting; no direct negative societal impact is identified.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The work does not release a high-risk generative model, scraped sensitive dataset, or other asset that requires controlled access safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Section 4 cites the creators of the datasets and baseline methods.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce a new dataset or benchmark asset; code and data access for reproducing experiments is addressed in the open-access checklist item above.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing experiments or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing or human subjects, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development does not use LLMs as an important, original, or non-standard component.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.