Compute-Efficient Event-Driven *Proceed* for Online Time Series Forecasting under Concept Drift

1st Juan Carlos La Rosa
Institute of Science and Technology
Federal University of São Paulo
São José dos Campos, São Paulo, 12247-014, Brazil
jclrparedes@unifesp.br

2nd Lilian Berton
Institute of Science and Technology
Federal University of São Paulo
São José dos Campos, São Paulo, 12247-014, Brazil
lberton@unifesp.br

***Abstract*—Online time series forecasting under concept drift requires models to adapt continuously as data distributions evolve. Recent proactive frameworks such as *Proceed* improve predictive accuracy by estimating drift and modulating model parameters before prediction, but this benefit comes at the cost of additional computation at every online step. In this work, we study efficiency-oriented variants of *Proceed* that aim to improve its accuracy–efficiency trade-off without changing the forecasting backbone. We evaluate an error-triggered variant, *Proceed*-ET, and a retrieval-based variant, *Proceed*-ER, as well as their combination. Experiments on ETTH2 and WEATHER with an *iTransformer* backbone show that *Proceed*-ET provides the clearest practical benefit: on WEATHER, it reduces total runtime from 1556.27 s to 870.15 s (-44.1%) and step latency from 33.30 to 16.20 ms/step (-51.4%), while maintaining accuracy close to the original *Proceed*; on ETTH2, it reduces runtime from 90.14 s to 52.02 s (-42.3%) and latency from 6.37 to 2.93 ms/step (-54.0%), although with a larger loss in accuracy. By contrast, *Proceed*-ER does not show consistent gains under the current configuration, and its benefits appear more plausible as a conditional rather than always-on component. Overall, our results suggest that selectively deciding *when* to adapt is a more effective lever for efficient proactive online forecasting than adding retrieval on top of constant adaptation, which is especially relevant in real-world settings with strict latency and memory constraints.**

***Index Terms*—Time series forecasting, concept drift, online learning, event-driven adaptation, transformers.**

I. INTRODUCTION

Time series forecasting (TSF) systems deployed in real environments rarely operate under a stationary data-generating process. Instead, the relationship between inputs and targets changes over time due to evolving conditions such as policies, sensors, or user behavior. This issue is closely related to *dataset shift* and, in streaming settings, to *concept drift* [1]–[3]. In online TSF, the problem is further complicated by temporal dependence and delayed supervision, since labels become available only after the forecast horizon [4], [5]. A common strategy for handling drift is to decide *when* to adapt using signals such as forecast error, distributional distance, or uncertainty [2], [6], [7]. However, deciding *when* to update is only part of the challenge, because effective online forecasting also requires deciding *how* to adapt under limited computational budgets. Many methods reduce drift to a scalar score or binary alarm, which offers little guidance about the type of update to apply. *Proceed* is a model adaptation framework for online time series forecasting that addresses this limitation by representing drift as a high-dimensional vector and using it to proactively modulate the forecasting model [5]. This idea is especially appealing under recurring drift, where previously observed regimes may reappear, and reusing past information can be preferable to relearning from scratch [8].

Recent online TSF methods reflect different trade-offs between adaptivity, stability, and computational cost. Detect-then-adapt approaches can react quickly, but may over-update when detection is noisy [4]. Ensemble and expert-based methods improve robustness at the cost of additional memory or computation [9]. Meta-learning and incremental-learning approaches seek efficient adaptation across evolving regimes, but often increase training complexity and sensitivity to shift structure [10], [11]. Meanwhile, transformer backbones have improved multivariate TSF performance, although strong forecasting accuracy alone does not guarantee robustness under non-stationarity [12]–[16]. These trade-offs motivate adaptation mechanisms that are not only accurate, but also compute-aware.

In this work, we study incremental modifications to *Proceed* to improve its efficiency in compute-constrained online forecasting. Rather than changing the forecasting backbone itself, we focus on lightweight extensions built on top of *Proceed*'s drift representation, with the goal of reducing the overhead of always-on proactive adaptation. This design choice allows us to evaluate whether efficiency gains can be achieved without giving up the main intuition of *Proceed*, namely, adapting the model according to an estimated representation of concept drift. Specifically, we compare these variants against both *Proceed (Original)* and a backbone-only baseline, and evaluate them in terms of predictive accuracy (MSE/MAE) as well as time-efficiency-related metrics such as total runtime and per-step latency.

The authors acknowledge the support of FAPESP (São Paulo Research Foundation) grants 2025/01895-9, 2021/14725-3, 2020/09850-0 and CNPq. The opinions, results, and conclusions expressed in this paper are those of the authors and do not necessarily reflect the views of FAPESP.

II. Related Work

We review prior work on online TSF under distribution shift, memory-based adaptation under recurring drift, and efficient event-triggered adaptation.

A. Online TSF under distribution shift

Moreno-Torres *et al.* provide a useful conceptual framework for *dataset shift*, distinguishing different ways in which training and deployment distributions may differ [1]. In streaming settings, these mismatches are commonly studied under the notion of *concept drift*, which includes patterns such as abrupt, gradual, incremental, and recurring changes [2], [3]. In time series forecasting, adaptation is further complicated by delayed supervision, since labels become available only after the forecast horizon [4], [5].

From a continual-learning perspective, online adaptation also raises the stability–plasticity dilemma: models must remain plastic enough to incorporate new regimes without overwriting useful knowledge from past ones [17]. This tension is closely related to *catastrophic forgetting*, especially in nonstationary forecasting scenarios where regimes may recur over time. For this reason, methods that can selectively update the model or reuse stored past information are particularly relevant under recurring drift.

B. Memory-based adaptation under recurring drift

When drift is recurring, storing and reusing past information can improve both accuracy and efficiency [8]. Retrieval-augmented forecasting follows this idea by fetching historical segments that are relevant to the current context. For instance, RAFT retrieves similar historical candidates and leverages their associated futures to support prediction [18], while retrieval has also been combined with generative forecasting to guide prediction under complex temporal patterns [19].

Our work is related to this line of research, but focuses on *online adaptation* rather than only improving one-shot forecasts. Instead of learning a separate retrieval representation, we reuse *Proceed*'s drift-encoder embedding as the retrieval key and perform cosine-similarity top-K retrieval with temperature-scaled soft aggregation. This design keeps the mechanism lightweight while aligning retrieval with *Proceed*'s notion of drift directionality [5].

C. Efficient event-triggered adaptation

In online learning, adaptation mechanisms must often balance predictive performance with computational cost. Prior work has explored conditional computation, update skipping, and budgeted training as ways to reduce unnecessary computation in sequential models and continual-learning settings [20]–[22]. In drift-handling pipelines, update decisions are commonly triggered by online signals such as forecast error, distributional distances, or windowed statistics [2], [3]. These methods must be evaluated not only by accuracy, but also by false alarms and computational overhead [6], [7].

Our *Proceed*-ET follows this event-driven perspective by gating *Proceed*'s adaptation and retrieval calls through an error-threshold statistic, reducing redundant computation during stable periods. In parallel, *Proceed*-ER improves efficiency by reusing previously observed regimes through retrieval in *Proceed*'s drift space, without requiring a separate representation learner [5], [8].

III. Background: Key Components of *Proceed*

We briefly summarize the components of *Proceed* that are most relevant to our extensions: concept drift estimation, proactive parameter adaptation, and the per-step computations introduced by this mechanism [5].

A. Concept drift estimation

A central component of *Proceed* is its use of two concept encoders to compare the latest labeled training sample available at online time t (i.e., $D_t^- = (X_{t-H}, Y_{t-H})$) with the current test input X_t [5]. These encoders produce latent representations c_{t-H} and c_t:

$$c_{t-H} = E(X_{t-H}, Y_{t-H}), \qquad c_t = E'(X_t), \tag{1}$$

from which the drift representation is defined as

$$\delta_{t-H\to t} = c_t - c_{t-H}. \tag{2}$$

In the original paper, this high-dimensional representation is intended to capture both the degree and the direction of the concept drift, in contrast to scalar drift scores used only to decide when to adapt [5].

B. Proactive parameter adaptation

Given the drift representation $\delta_{t-H\to t}$ and the forecast model parameters θ_{t-H} after online fine-tuning on the latest available labeled sample, *Proceed* uses an adaptation generator with bottleneck layers to produce layer-wise adaptation coefficients [5]. For the ℓ-th layer, the generator outputs coefficients $\alpha_t^{(\ell)}$ and $\beta_t^{(\ell)}$, which are used to define the adapted parameters as

$$\hat{\theta}_t^{(\ell)} = \left(\alpha_t^{(\ell)\top}\beta_t^{(\ell)}\right) \odot \theta_{t-H}^{(\ell)}, \tag{3}$$

where $\odot$ denotes element-wise multiplication [5]. In this sense, *Proceed* does not directly generate a new full parameter tensor, but instead predicts a compact structured rescaling of the existing parameters.

At the solution-overview level, *Proceed* applies this adaptation for the current online prediction and resets the parameters at the next step [5]. The appendix further notes that the deployed online pipeline is slightly more involved in order to remain consistent with joint training, while the model adapter itself remains frozen during the online phase [5].

C. Per-step computational overhead

Because *Proceed* estimates drift and invokes the adaptation generator at each online step, it introduces additional computation beyond the backbone forecast alone. The original paper addresses this cost through a bottleneck generator and provides time-complexity analyses for both concept drift estimation and proactive adaptation [5]. In our setting, this repeated per-step overhead motivates studying more selective adaptation

mechanisms that preserve the main idea of *Proceed* while reducing unnecessary computation under stable regimes.

IV. Proposed Methods

We build on *Proceed*, which estimates concept drift between the latest labeled context and the current input, and uses this signal to modulate the forecasting model before prediction. Our goal is to improve its accuracy–efficiency trade-off without changing the forecasting backbone. To this end, we introduce two lightweight extensions: *Proceed*-ET, which selectively triggers adaptation based on recent error, and *Proceed*-ER, which reuses previously observed contexts through retrieval.

A. Proceed-*ET: Error-Triggered Adaptation*

The original *Proceed* pipeline performs drift estimation and proactive adaptation at every online step. *Proceed*-ET makes this process event-driven: adaptation is executed only when recent delayed error indicates that the current regime is difficult enough to justify the extra computation.

Because supervision arrives after the forecast horizon, the error available at time t corresponds to the prediction made at time $t - H$. We define the delayed scalar error as

$$e_t = \frac{1}{dH} \left\| \hat{Y}_{t-H} - Y_{t-H} \right\|_2^2, \tag{4}$$

where $\hat{Y}_{t-H}, Y_{t-H} \in \mathbb{R}^{H \times d}$.

To reduce sensitivity to short-term fluctuations, we optionally smooth this signal and compute an adaptive threshold from a recent window of errors. Let $p \in (0, 1]$ denote `adapt_top_p`. We set

$$\theta_t = \mathrm{Quantile}_{1-p}\big(\bar{e}_{t-w}, \ldots, \bar{e}_{t-1}\big), \tag{5}$$

and define the gate as

$$g_t = \mathbb{I}(\bar{e}_t > \theta_t). \tag{6}$$

In this way, adaptation is triggered only on the highest-error fraction of recent steps, making the compute budget directly controllable through p. During an initial warm-up period, we set $g_t = 1$ so that the method behaves as standard *Proceed* until enough delayed errors are available.

When $g_t = 0$, we skip proactive adaptation and forecast using the most recent fine-tuned model state. When $g_t = 1$, we execute the full *Proceed* pipeline. Thus, *Proceed*-ET preserves the original adaptation mechanism while avoiding unnecessary updates during stable periods.

B. Proceed-*ER: Embedding Retrieval*

Proceed-ER introduces a lightweight memory bank to reuse previously observed contexts. The intuition is that, under recurring regimes, a retrieved past reference can provide a more stable basis for drift estimation than relying only on the most recent labeled context.

Let $c_t \in \mathbb{R}^D$ denote the current concept embedding and let $r_t \in \mathbb{R}^D$ denote the recent reference embedding derived from the latest labeled context. For retrieval, we use the normalized current embedding as query,

$$q_t = \frac{c_t}{\|c_t\|_2}. \tag{7}$$

Given a memory bank of normalized past queries $\mathcal{M} = \{m_i\}_{i=1}^n$, we compute cosine similarities $s_i = m_i^\top q_t$, retrieve the top-K neighbors, and aggregate them with temperature-scaled softmax weights:

$$w_i = \frac{\exp(s_i/\tau)}{\sum_{j \in \mathcal{N}_t} \exp(s_j/\tau)}, \qquad \tilde{m}_t = \frac{\sum_{i \in \mathcal{N}_t} w_i m_i}{\left\| \sum_{i \in \mathcal{N}_t} w_i m_i \right\|_2}. \tag{8}$$

We then convert the retrieved direction into a retrieved reference with the same scale as the recent reference,

$$\tilde{r}_t = \|r_t\|_2 \, \tilde{m}_t, \tag{9}$$

and combine both sources of information as

$$\mathrm{ref}_t = \alpha r_t + (1 - \alpha)\tilde{r}_t, \tag{10}$$

where $\alpha \in [0, 1]$ corresponds to `retrieval_alpha`. Finally, drift is computed as

$$\delta_t = c_t - \mathrm{ref}_t, \tag{11}$$

and passed to the same adaptation generator used by *Proceed*.

This design keeps retrieval lightweight: it does not introduce a separate representation learner, and it only modifies how the drift reference is formed. To avoid leakage, the current query is inserted into the memory bank only after retrieval at time t.

C. Proceed-*ET+ER*

The combined model uses *Proceed*-ET to decide *when* adaptation should be executed, and *Proceed*-ER to refine *how* drift is estimated whenever adaptation is triggered. In this way, ET targets computational efficiency, while ER targets robustness under recurring regimes.

V. Experimental Setup

A. Datasets and online evaluation protocol

We follow the online TSF setting of *Proceed*, where supervision for a prediction becomes available only after the forecast horizon, and evaluation is performed over an online test stream [5]. We consider two multivariate datasets, ETTh2 and Weather, and use multivariate-to-multivariate forecasting (`features=M`) throughout.

For both datasets, we set the lookback length to $L = 96$ and the forecast horizon to $H = 96$. The label length in the decoder input is set to 48, following the original configuration. To avoid leakage, all data splits are chronological and non-shuffled, with the standard overlap induced by the lookback window when constructing the online stream.

a) ETTh2*:* Following the *Proceed* `border_type=online` setup for ETT datasets, we use a chronological partition of 50% for training, 16.67% for validation, and 33.33% for testing.

TABLE I
DATASET CONFIGURATION USED IN OUR EXPERIMENTS.

Dataset	Variables (d)	Lookback (L)	Horizon (H)
ETTh2	7	96	96
Weather	21	96	96

b) WEATHER*:* Following the original custom-dataset split used in *Proceed*, we adopt a chronological partition of 70% for training, 10% for validation, and 20% for testing.

B. Compared methods

We compare the original *Proceed*, a backbone-only baseline, and our proposed variants.

a) ***iTransformer (No Proceed)****:* This backbone-only baseline uses the same pretrained forecasting model but removes *Proceed*'s drift representation, retrieval, and proactive adaptation pipeline. Online adaptation is performed only through standard gradient-based updates on each arriving labeled batch.

b) ***Proceed (Original)****:* This is the original proactive adaptation framework, which estimates concept drift and uses a generator to modulate model parameters before prediction [5].

c) ***Proceed****-ET:* This variant introduces *error-triggered adaptation*: the proactive adaptation pipeline is executed only when the delayed online error exceeds an adaptive threshold derived from recent history.

d) ***Proceed****-ER:* This variant introduces *embedding retrieval*: a memory bank of past embeddings is queried to form a refined reference for drift estimation under recurring regimes.

e) ***Proceed****-ET+ER:* This combined variant uses ET to decide *when* to adapt and ER to refine *how* drift is estimated when adaptation is triggered.

Across all *Proceed*-based variants, the forecasting backbone and the original *Proceed* components are kept fixed; only the triggering and retrieval logic differs.

C. Backbone and training protocol

To ensure comparability across methods, we use **iTransformer** as the forecasting backbone in all experiments [16]. We follow the original *Proceed* training recipe: the model is first pretrained for up to 10 epochs with early stopping (patience $= 3$), after which online evaluation is performed on the test stream according to the corresponding method [5].

D. Hyperparameter selection

We keep all backbone and original *Proceed* hyperparameters fixed and tune only the additional parameters introduced by our variants using Optuna. Specifically, we tune: `adapt_top_p` for *Proceed*-ET, and `retrieval_alpha` and `tau` for *Proceed*-ER.

We use a two-stage tuning procedure. In Stage 1, Optuna minimizes validation MSE. In Stage 2, it minimizes total runtime subject to an accuracy constraint. We penalize any candidate configuration such that

$$\mathrm{MSE}_{\mathrm{val}} > 1.01 \times \mathrm{MSE}_{\mathrm{val}}^{\mathrm{baseline}}, \tag{12}$$

TABLE II
OPTUNA-SELECTED HYPERPARAMETERS USED IN THE FINAL EXPERIMENTS.

Dataset	`adapt_top_p`	`retrieval_alpha`	`tau`
ETTh2	0.1332	0.8191	0.132209
Weather	0.3944	0.6128	0.004701

where $\mathrm{MSE}_{\mathrm{val}}^{\mathrm{baseline}}$ is the validation MSE of ***Proceed*** **(Original)** under the same protocol. This enforces our main design goal: improved efficiency with at most 1% relative degradation in validation accuracy.

Unless explicitly tuned, we use the default infrastructure settings from the codebase: memory bank size `bank_size`=2048, top-K $=$ 8, gating window `gate_window`=256, and an initial warm-up period `warmup_steps` during which triggering is disabled.

E. Evaluation metrics

We report predictive accuracy using MSE and MAE. To measure efficiency in the online setting, we report total runtime per dataset and seconds per online step, as recorded by the evaluation loop.

To verify the behavior of *Proceed*-ET, we also report *% updates*, defined as the percentage of online steps in which a backward pass and `optimizer.step()` are executed. For retrieval-based variants (*Proceed*-ER and *Proceed*-ET+ER), we additionally report *% retrieval*, i.e., the percentage of total runtime spent in retrieval operations. These statistics are important because online drift-handling methods may incur substantial overhead when update decisions are triggered too frequently by noisy signals [6], [7].

F. Reproducibility and implementation details

Each configuration is run with three repetitions (`itr`=3), and we report mean $\pm$ standard deviation over the online test stream. We follow the codebase's deterministic seeding scheme, where the random seed is offset by the repetition index.

All experiments were executed on a personal machine running macOS Sonoma 14.6.1 with an Apple M1 CPU and 8 GB RAM, using CPU-only execution. The codebase was run in a Python virtual environment managed with `venv`/`pip`.

VI. RESULTS

Tables III, IV, and V report accuracy (MSE/MAE) and efficiency (total runtime, ms/step, and `% updates`) for *Proceed* and the proposed variants. For retrieval-based variants (*Proceed*-ER and *Proceed*-ET+ER), we additionally report `% retrieval`, defined as the fraction of total runtime spent in retrieval.

On WEATHER (Tables III and IV), *Proceed*-ET and *Proceed*-ET+ER reduce total runtime from 1556.27 s to 870.15 s and 850.14 s (-44.1% and -45.4%), and reduce step latency from 33.30 ms/step to 16.20 ms/step and 15.70 ms/step (-51.4% and -52.9%). In this dataset, their MSE/MAE are close to the *Proceed* baseline (2.47/0.84), with *Proceed*-ET

TABLE III
FORECASTING ACCURACY (MEAN±STD OVER ITR=3)

Method	WEATHER		ETTH2	
	MSE	MAE	MSE	MAE
iTransformer (No *Proceed*)	2.67±0.05	0.93±0.02	6.40±0.28	0.94±0.02
Proceed (Original)	2.47±0.34	0.84±0.06	5.83±0.13	0.91±0.01
Proceed-ET	2.41±0.11	0.83±0.02	6.35±0.17	0.93±0.01
Proceed-ER	2.56±0.25	0.86±0.04	5.82±0.14	0.91±0.01
Proceed-ET+ER	2.37±0.20	0.84±0.04	6.35±0.17	0.93±0.01

TABLE IV
EFFICIENCY ON WEATHER (MEAN±STD OVER ITR=3)

Method	Total time (s)	ms/step	% updates	% retrieval
iTransformer (No *Proceed*)	790.60±21.75	16.40±0.44	100.00±0.00	–
Proceed (Original)	1556.27±68.63	33.30±1.51	100.00±0.00	–
Proceed-ET	870.15±48.75	16.20±0.96	39.15±0.86	–
Proceed-ER	1649.99±38.66	35.27±0.85	100.00±0.00	2.49±0.03
Proceed-ET+ER	850.14±19.10	15.70±0.36	39.08±1.21	2.79±0.05

obtaining 2.41/0.83 and *Proceed*-ET+ER obtaining 2.37/0.84. By contrast, *Proceed*-ER increases both runtime (1649.99 s; $+6.0\%$) and step latency (35.27 ms/step; $+5.9\%$), while also yielding higher error (2.56/0.86).

On ETTH2 (Tables III and V), *Proceed*-ET and *Proceed*-ET+ER again reduce runtime (90.14 s to 52.02 s and 62.85 s; -42.3% and -30.3%) and latency (6.37 ms/step to 2.93 ms/step and 3.57 ms/step; -54.0% and -44.0%), but these gains coincide with higher MSE/MAE (6.35/0.93 for both) compared to *Proceed* (5.83/0.91). *Proceed*-ER matches *Proceed* more closely in accuracy on ETTH2 (5.82/0.91), at the cost of a modest runtime increase (94.64 s; $+5.0\%$) and higher latency (6.63 ms/step; $+4.1\%$). For retrieval-based methods, `% retrieval` remains low, ranging from 0.94% (ETTh2, *Proceed*-ER) to 2.79% (Weather, *Proceed*-ET+ER).

VII. ANALYSIS AND DISCUSSION

We analyze the cost–accuracy trade-off of *Proceed* relative to its non-proactive backbone (*iTransformer*, No *Proceed*), focusing on two mechanisms: error-triggered gating (ET), which controls *when* adaptation runs, and embedding retrieval (ER), which affects *which* past contexts are emphasized.

A. ET reduces overhead, but its effect on accuracy is dataset-dependent

Across both datasets, *Proceed* improves accuracy over *iTransformer* but increases runtime and latency substantially (Tables III, IV, and V). ET reduces this overhead by lowering `% updates` from 100% to about 39% on WEATHER and 24% on ETTH2, bringing ms/step close to the backbone-only baseline. This suggests that update frequency is a major source of cost in *Proceed*-style pipelines.

Its effect on accuracy, however, depends on the stream. On WEATHER, ET remains competitive with *Proceed* while greatly reducing cost. On ETTH2, ET moves closer to the backbone-only baseline and does not retain the lower mean error of *Proceed*. A plausible explanation is that delayed error-based triggering may work well when many updates are redundant, but may be too conservative when drift is frequent or subtle.

TABLE V
EFFICIENCY ON ETTH2 (MEAN±STD OVER ITR=3)

Method	Total time (s)	ms/step	% updates	% retrieval
iTransformer (No *Proceed*)	37.72±0.86	2.77±0.06	100.00±0.00	–
Proceed (Original)	90.14±1.10	6.37±0.06	100.00±0.00	–
Proceed-ET	52.02±0.49	2.93±0.06	23.79±0.34	–
Proceed-ER	94.64±2.37	6.63±0.15	100.00±0.00	0.94±0.02
Proceed-ET+ER	62.85±0.87	3.57±0.06	23.77±0.21	1.40±0.04

A practical implication of these results is that the value of *Proceed*-ET is not limited to reducing wall-clock time. In online forecasting systems, adaptation must often operate under strict latency and compute constraints, especially when predictions are updated continuously or deployed on limited hardware. In such settings, a method that preserves competitive accuracy while reducing runtime by more than 40% and step latency by around 50% can make proactive adaptation substantially more viable in practice. From this perspective, *Proceed*-ET is not only a lighter variant of *Proceed*, but also a step toward making drift-aware online forecasting more deployable in real-world environments.

B. ER shows limited benefit under the current configuration

ER alone does not provide consistent gains over *Proceed*. It increases runtime on both datasets and does not show a reliable accuracy improvement: it is worse on WEATHER and nearly tied with *Proceed* on ETTH2. Since `% updates` remains at 100%, these results suggest that retrieval overhead is secondary compared with the cost of always-on adaptation.

C. ET+ER is more plausible than ER alone, but evidence is still limited

Proceed-ET+ER applies retrieval only when ET triggers an update. On WEATHER, it achieves the lowest reported mean MSE among the evaluated methods while keeping a runtime close to *Proceed*-ET. However, given the reported standard deviations, this should be interpreted as a promising trend rather than conclusive superiority. On ETTH2, ET+ER matches ET in mean accuracy while remaining slower, which weakens the case for retrieval in this setting.

Overall, ET appears to be the main source of efficiency gains, whereas ER is more plausible as a conditional component than as an always-on mechanism.

D. Limitations and future work

The efficiency pattern is clear, but the accuracy comparisons should be interpreted with caution. Since results are reported as mean±std over only `itr`=3 runs, we do not make strong claims of statistical superiority when differences are small.

Future work should include more repetitions, confidence intervals, formal statistical tests, and Pareto-style analysis of the accuracy–efficiency frontier. Step-wise analyses of update decisions, drift intensity, and retrieval usefulness would also help explain when selective adaptation is most effective.

VIII. Conclusion

In this work, we studied efficiency-oriented variants of *Proceed* for online time series forecasting under concept drift. Rather than redesigning the original framework, our goal was to reduce the overhead of always-on proactive adaptation while preserving its predictive benefits as much as possible.

Among the proposed variants, *Proceed*-ET provides the clearest improvement in the accuracy–efficiency trade-off. By triggering adaptation only when recent delayed error indicates it is necessary, *Proceed*-ET reduces total runtime by more than 40% and step latency by around 50% across both datasets. On WEATHER, these savings are achieved while maintaining accuracy close to the original *Proceed*, whereas on ETTH2 the same efficiency gains come with a larger loss in accuracy. This shows that selective adaptation can substantially improve practicality, although the final trade-off remains stream-dependent.

By contrast, *Proceed*-ER does not provide consistent benefits under the current configuration. While retrieval may still be useful as a conditional component, its always-on version adds cost without showing reliable accuracy gains. Overall, our findings suggest that deciding *when* to adapt is a more effective lever than refining adaptation with retrieval alone.

This is especially relevant for real-world online systems, where memory and latency constraints can make adaptation at every step impractical. In such settings, reducing computation by roughly 40–50% while preserving competitive accuracy can make proactive adaptation considerably more viable. Future work should extend this analysis with broader dataset coverage, stronger statistical evaluation, and Pareto-style comparisons to better characterize the accuracy–efficiency frontier of selective adaptation.

Declaration of AI in the writing process

During the preparation of this work, the authors used the ChatGPT LLM to assist with technical writing, structural refinement and grammar checking. The authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Code Availability

The implementation used in this work is publicly available at https://github.com/juanlarosap/Compute_Efficient_Proceed.

References

[1] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320311002901

[2] J. a. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014. [Online]. Available: https://doi.org/10.1145/2523813

[3] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2876857

[4] Y. Zhang, W. Chen, Z. Zhu, D. Qin, L. Sun, X. Wang, Q. Wen, Z. Zhang, L. Wang, and R. Jin, "Addressing Concept Shift in Online Time Series Forecasting: Detect-then-Adapt," Mar. 2024, arXiv:2403.14949 [cs]. [Online]. Available: http://arxiv.org/abs/2403.14949

[5] L. Zhao and Y. Shen, "Proactive model adaptation against concept drift for online time series forecasting," p. 2020–2031, 2025. [Online]. Available: https://doi.org/10.1145/3690624.3709210

[6] P. M. Gonçalves, S. G. de Carvalho Santos, R. S. Barros, and D. C. Vieira, "A comparative study on concept drift detectors," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8144–8156, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417414004175

[7] M. D. L. Tosi and M. Theobald, "OPTWIN: Drift Identification with Optimal Sub-Windows," in *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. Utrecht, Netherlands: IEEE, May 2024, pp. 331–337. [Online]. Available: https://ieeexplore.ieee.org/document/10555081/

[8] A. L. Suárez-Cetrulo, D. Quintana, and A. Cervantes, "A survey on machine learning for recurring concept drifting data streams," *Expert Systems with Applications*, vol. 213, p. 118934, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422019522

[9] Y.-F. Zhang, Q. Wen, X. Wang, W. Chen, L. Sun, Z. Zhang, L. Wang, R. Jin, and T. Tan, "OneNet: Enhancing Time Series Forecasting Models under Concept Drift by Online Ensembling," Sep. 2023, arXiv:2309.12659 [cs]. [Online]. Available: http://arxiv.org/abs/2309.12659

[10] L. Zhao, S. Kong, and Y. Shen, "DoubleAdapt: A Meta-learning Approach to Incremental Learning for Stock Trend Forecasting," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Long Beach CA USA: ACM, Aug. 2023, pp. 3492–3503. [Online]. Available: https://dl.acm.org/doi/10.1145/3580305.3599315

[11] W. Chen, Z. Zhu, Y. Zhang, L. Shen, L. Yang, Q. Wen, and L. Sun, "Learning to extrapolate and adjust: two-stage meta-learning for concept drift in online time series forecasting," 2025. [Online]. Available: https://doi.org/10.24963/ijcai.2025/542

[12] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are Transformers Effective for Time Series Forecasting?" Aug. 2022, arXiv:2205.13504 [cs]. [Online]. Available: http://arxiv.org/abs/2205.13504

[13] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11 106–11 115, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17325

[14] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting," Jan. 2022, arXiv:2106.13008 [cs]. [Online]. Available: http://arxiv.org/abs/2106.13008

[15] B. Lim, S. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021000637

[16] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "iTransformer: Inverted Transformers Are Effective for Time Series Forecasting," Mar. 2024, arXiv:2310.06625 [cs]. [Online]. Available: http://arxiv.org/abs/2310.06625

[17] Q. Besnard and N. Ragot, "Continual Learning for Time Series Forecasting: A First Survey," in *ITISE 2024*. MDPI, Jul. 2024, p. 49. [Online]. Available: https://www.mdpi.com/2673-4591/68/1/49

[18] S. Han, S. Lee, M. Cha, S. O. Arik, and J. Yoon, "Retrieval Augmented Time Series Forecasting," May 2025, arXiv:2505.04163 [cs]. [Online]. Available: http://arxiv.org/abs/2505.04163

[19] J. Liu, L. Yang, H. Li, and S. Hong, "Retrieval-Augmented Diffusion Models for Time Series Forecasting," Oct. 2024, arXiv:2410.18712 [cs]. [Online]. Available: http://arxiv.org/abs/2410.18712

[20] A. Graves, "Adaptive computation time for recurrent neural networks," *CoRR*, vol. abs/1603.08983, 2016. [Online]. Available: https://arxiv.org/abs/1603.08983

[21] V. Campos, B. Jou, X. Giro-i Nieto, J. Torres, and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," *CoRR*, vol. abs/1708.06834, 2017. [Online]. Available: https://arxiv.org/abs/1708.06834

[22] M. Li, E. Yumer, and D. Ramanan, "Budgeted training: Rethinking deep neural network training under resource constraints," 2020. [Online]. Available: https://arxiv.org/abs/1905.04753