Energy Efficient Streaming Time Series Classification with Attentive Power Iteration

Hao Huang[1], **Tapan Shah**[1], **Scott Evans**[1], **Shinjae Yoo**[2]
[1]GE Vernova Research, Niskayuna, NY, USA
[2]Brookhaven National Lab, Upton, NY, USA
hao.huang1@ge.com, tapan.shah@ge.com, evans@ge.com, sjyoo@bnl.gov

Abstract

Efficiently processing time series data streams in real-time on resource-constrained devices offers significant advantages in terms of enhanced computational energy efficiency and reduced time-related risks. We introduce an innovative streaming time series classification network that utilizes attentive power iteration, enabling real-time processing on resource-constrained devices. Our model continuously updates a compact representation of the entire time series, enhancing classification accuracy while conserving energy and processing time. Notably, it excels in streaming scenarios without requiring complete time series access, enabling swift decisions. Experimental results show that our approach excels in classification accuracy and energy efficiency, with over 70% less consumption and threefold faster task completion than benchmarks. This work advances real-time responsiveness, energy conservation, and operational effectiveness for constrained devices, contributing to optimizing various applications.

Introduction

Resource-constrained devices, characterized by limited computational capacity and memory, are essential components of modern infrastructures. Devices such as edge devices (e.g., routers, smartphones) and distributed energy resources (DERs) like solar PV units, turbines, and gas units, all fall within this category. These devices accumulate vast quantities of high-frequency time series data streams over time, rendering manual real-time analysis impractical. Artificial Intelligence (AI) models, particularly those operating on time series data, have gained traction due to their ability to autonomously discern underlying patterns.

However, existing AI models, particularly popular neural networks, often demand substantial memory and computational energy, rendering them unsuitable for direct deployment on resource-constrained environment. A common workaround is to transmit collected data to cloud servers for analysis, but this approach consumes bandwidth, time, and raises concerns about data privacy and security during transmission. Moreover, many AI models rely on batch learning, which processes entire time series simultaneously, instead of incremental learning required for real-time analysis.

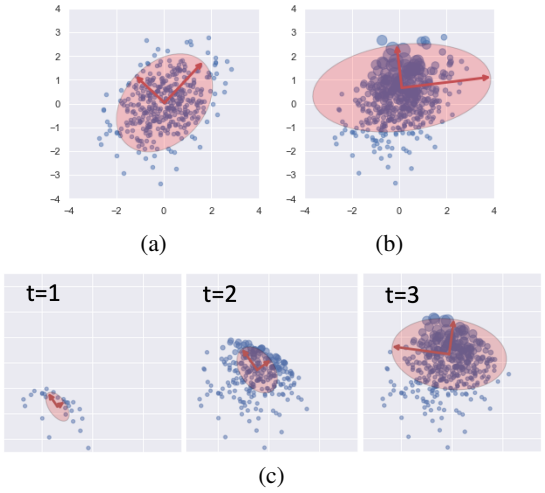
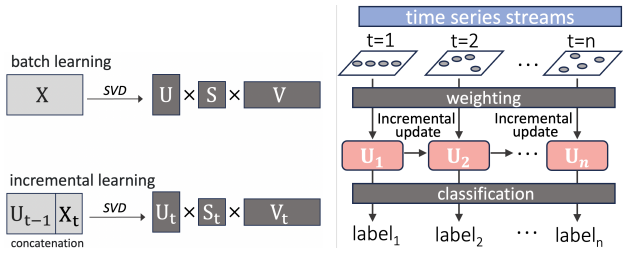


Figure 1: Fig 1(a) showcases regular PCA on a 2D dataset, while Fig 1(b) enhances this with weighted samples (sample size denotes weight). In Fig 1(c), we introduce an incremental weighted PCA within a supervised neural network's latent space. Samples' weights, determined by an attention model, capture their significance within the time series. The real-time principal direction resulting from this process becomes a distinct representation for real-time classification.

To address these challenges, there is a pressing need to develop streaming and cost-effective models capable of real-time analysis of time series data streams directly on resource-constrained environment. This study specifically targets the streaming time series classification problem, where categorical class labels are predicted for each time series consisting of ordered sets of real-valued attributes, often multivariate. Streaming time series classification operates on time series streams, generating real-time labels while processing the data only once and utilizing limited storage.

We propose a neural network for streaming time series classification, engineered to operate with restricted computational memory and energy resources, while ensuring high processing speed. In our approach, as data from the same time series arrives sequentially, we construct a real-time representation using an incremental weighted Principal Com-



(a) (b)

Figure 2: Fig 2(a) depicts data reduction through conventional batch/incremental learning. In contrast, Fig 2(b) introduces our pioneering incremental learning schema, wherein the time series sketch U experiences incremental updates via weighted time series batches, ultimately producing a real-time representation for streaming classification.

ponent Analysis (PCA) on a supervised neural network-learned latent space. This representation is then employed for time series classification. While traditional PCA (Figure 1(a)) is an unsupervised method for dimensionality reduction in non-time series data, we apply it to a temporally and supervisedly learned latent space to accommodate the nature of time series data. Since time series samples are not equally significant, we introduce weighted PCA, depicted in Figure 1(b), where each sample is assigned weights using a temporal attention model. With such weighted time series streams, our model incrementally updates principal directions, forming distinctive real-time representations for classification as shown in Figure 1(c). The complete architecture, encompassing latent space learning and incremental weighted PCA, is trained end-to-end in a supervised learning manner. Specifically, our model utilizes **str**eaming **A**ttentive **P**ower **I**teration to incrementally refine principal directions within the supervised latent space. We term our proposed model ***strAPI***. *StrAPI* generates real-time representations and labels for each time series without needing to observe the entire sequence. Experimental results reveal that our *strAPI* algorithm consistently achieves superior classification accuracy with minimal energy consumption and faster computational speed, on average, when compared against popular streaming baselines in time series classification.

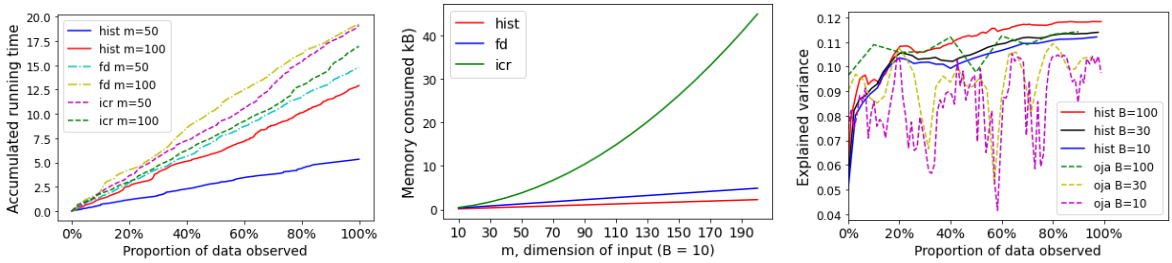
Motivations, Related Work and Notations

Related Work. Due to the inherent limitations of memory in resource-constrained environment, traditional time series analysis like FFT (Bloomfield 2004), Random Forest (Tyralis and Papacharalampous 2017), and Dynamic Time Warping (Jeong, Jeong, and Omitaomu 2011) are not suitable in this context since they require full access to the whole time series. Numerous research endeavors have been undertaken to derive a concise and updatable representation from observed data, tailored for streaming analysis (Coleman and Shrivastava 2020; Kowshik et al. 2021; Jin et al. 2021; Chen et al. 2022; Diakonikolas et al. 2022; Makarychev, Manoj, and Ovsiankin 2022). Among these approaches, *matrix sketching* has emerged as a viable solution, finding application in diverse streaming scenarios such as anomaly detection (Huang and Kasiviswanathan 2015), clustering (Yoo, Huang, and Kasiviswanathan 2016), visualization (Fujiwara et al. 2019) and non-time series classifications (Falcone, Anderlucci, and Montanari 2022). A matrix sketch is a reduced-size representation of an input matrix that effectively retains its essential characteristics. This technique expedites numerical analysis on extensive data streams, rendering it more efficient in terms of both processing speed and memory utilization. Concurrently, recurrent neural networks (RNNs) (Bengio, Simard, and Frasconi 1994; Salehinejad et al. 2017; Mohammadi et al. 2018; Dennis et al. 2019) also excel in maintaining a succinct representation of observed time series data. The architecture of RNNs incorporates hidden states that function as the memory repository of the network (Salehinejad et al. 2017). The state of the hidden layer at a given moment is contingent on its previous state, enabling RNNs to encapsulate, retain, and process the entirety of observed data streams within a compact representation. However, prevalent methods of recurrence in RNNs can engender an excessive number of model parameters, leading to sluggish training and inference speeds (Dennis et al. 2019). Moreover, RNNs contend with the challenge of catastrophic forgetting issues, which impede their ability to effectively learn from extended time series sequences (Van Houdt, Mosquera, and Nápoles 2020; Ribeiro et al. 2020).

Motivation. Our proposed *strAPI* finds its foundation in matrix sketching methodologies. As depicted in Figure 2(a), the conventional data reduction process involves constructing a linear model that maps the original input matrix $X \in R^{m\times n}$ to a lower-dimensional latent factor $U \in R^{m\times k}$, with $k \ll n$ (Ross et al. 2008; Liberty 2013). Distinct from the batch approach of data reduction that processes the entire dataset, matrix sketching operates on individual samples or limited sample batches, updating the sketch U to encapsulate all observed data up to that point.

StrAPI represents the pioneering integration of matrix sketching techniques within a neural network framework, specifically tailored for real-time time series classification. Illustrated in Figure 2(b), our approach involves incorporating unsupervised matrix sketching within a supervised context. This integration enables the continuous updating of the sketch linked to a sequence of weighted time series streams. The dynamically updated sketch serves as a compact representation of the input time series, facilitating real-time classification into known time series classes at any given timestamp. This innovative amalgamation of unsupervised matrix sketching with supervised neural networks addresses the demands of real-time processing while maintaining the accuracy and efficiency of classification tasks.

Notation and Problem Setting. A time series comprising m variables and n timestamps is represented as $X \in R^{m\times n}$. We assume that each entire time series belongs to a single class. **Our task** involves a set of time series $X^1, X^2, ..., X^N$ (N denotes the total number of time series) with consistent variable count m but possibly differing lengths. Our goal is to develop a classification algorithm that assigns a class label $label_t^i$ to each time series X^i at any timestamp t, based on the observed part $X_{1:t}^i$, in a streaming manner.



(a) (b) (c)

Figure 3: Figures 3(a) and 3(b) depict runtime and memory usage comparisons of hist-PCA (Yang et al. 2018) against decomposition-based methods (icr and fd for incremental-PCA (Ross et al. 2008) and frequent directions (Liberty 2013)). Figure 3(c) illustrates explained variance by hist-PCA and Oja's method (Oja and Karhunen 1985), where m signifies sample dimensions and B represents time series batch size. We can see that hist-PCA is more efficient and effective within our context.

Our Design Methodology

Choose a Suitable Way for Matrix Sketching

Matrix sketching can be broadly categorized into two prevalent types: *decomposition-based methods* and *power methods*. Decomposition-based methods involve updating the matrix sketch through processes like eigen-decomposition or singular value decomposition (SVD), particularly when new samples are introduced (depicted in Figure 2(a)). Examples include incremental PCA (Cardot and Degras 2018; Chen et al. 2020) and Frequent Directions-based methods (Liberty 2013; Yi et al. 2022). However, decomposition-based methods pose certain challenges:

1. **Complexity Challenges**. Decomposition-based methods often exhibit substantial time and space complexity, which can pose difficulties for real-time analysis on resource-constrained devices. Figure 3(a) and 3(b) demonstrate a comparative analysis of running times and memory consumption. Notably, power methods like hist-PCA (Yang et al. 2018) outperform decomposition-based alternatives, such as incremental PCA and Frequent Direction, in terms of both speed and memory efficiency. This superiority is particularly pronounced when dealing with high-dimensional input data (dimension m), rendering power methods more practical for real-time streaming analysis within memory-limited environments.
2. **Ill-Conditioning Issues**. Recent research has highlighted potential pitfalls when integrating matrix decomposition into neural networks. This integration can lead to ill-conditioned covariance matrices, characterized by nearly equal or repeated eigenvalues. Such ill-conditioning can destabilize model training and hinder generalization capabilities (Song et al. 2022b). The presence of trivially-small eigenvalues during network training's early stages, combined with the stochastic nature of gradient descent, can induce inaccuracies during eigenvalue and eigenvector estimation. These inaccuracies result in large round-off errors, potentially leading to numerical instability or runtime errors (Song et al. 2022a).

Therefore, **power methods emerge as a more suitable option within our context**. Unlike classical power methods (Lin and Cohen 2010; Zhang et al. 2022) that iteratively multiply the estimated sketch vector with the entire data covariance matrix $U_i \leftarrow \frac{1}{n} X X^T U_{i-1}$, our preference shifts towards the batch stochastic power method (Oja and Karhunen 1985; Mitliagkas et al. 2013). This variant updates the current estimate in a streaming manner by

$$U_i \leftarrow \frac{1}{B} X_i ({X_i}^T U_{i-1}) \tag{1}$$

with $X_i \in R^{m \times B}$ representing the new arriving batch at time i with B samples. The order of computation obviates the need to explicitly form the covariance matrix, making it memory-efficient, especially for large input dimensions (m).

For Equation 1, it has been proved that using a large batch size can reduce the potentially large variance (Oja and Karhunen 1985; Mitliagkas et al. 2013; Yang et al. 2018). However, *our setting precludes accommodating large batches due to memory limitations or real-time analytical demands*. This makes stability imperative, even with small batch inputs. Therefore hist-PCA (Yang et al. 2018) is harnessed in our approach, as described in Algorithm 1. Unlike Equation 1, hist-PCA progressively incorporates the knowledge gained from new covariances into the latest sketch until convergence, as demonstrated in lines 4 and 10 of Algorithm 1. Empirical evidence presented in Figure 3(c) confirms hist-PCA's efficacy at an early stage and its ability to maintain stable estimates over time, even with small batches.

Algorithm 1's intuition stems from its iterative nature. Upon the arrival of the first batch, for convergence convenience, the iteration occurs on $I + \frac{1}{B} X_1 {X_1}^T$ rather than $\frac{1}{B} X_1 {X_1}^T$ alone. With the second batch's introduction, an ideal scenario involves formulating the sample covariance matrix as $\frac{1}{2B}(X_1 {X_1}^T + X_2 {X_2}^T)$, assuming both batches are in memory. However, given memory constraints inherent in resource-constrained devices, an alternative approach involves incorporating essential insights from the latest sketch, embodied in $\frac{1}{2} U_1 U_1^T + \frac{1}{2B} X_2 {X_2}^T$. Line 10 of Algorithm 1 delineates this process in a general format, where $\frac{i-1}{i}$ and $\frac{1}{i}$ represent weights attributed to the observed data volume for the latest sketch covariance and the new batch covariance, respectively (i denotes the batch index).

Algorithm 1: hist-PCA (Yang et al. 2018)

Input: a sequence of data batch $\{X_1, ..., X_n\}$, batch size B, starting vector $U_0 \sim N(0, I_{m\times m})$
Output: final sketch U_n

Initialization : $U_1 \leftarrow U_0/\|U_0\|_2$
1: **for** $i = 1$ to n **do**
2: **if** $i = 1$ **then**
3: **while** U_1 not converge **do**
4: $U_1 \leftarrow U_1 + \frac{1}{B} X_1 X_1^T U_1$
5: $U_1 \leftarrow U_1/\|U_1\|_2$
6: **end while**
7: **else**
8: $U_i \leftarrow U_{i-1}$
9: **while** U_i not converge **do**
10: $U_i \leftarrow \frac{i-1}{i} U_{i-1} U_{i-1}^T U_i + \frac{1}{i}\frac{1}{B} X_i X_i^T U_i$
11: $U_i \leftarrow U_i/\|U_i\|_2$
12: **end while**
13: **end if**
14: **end for**
15: **return** U_n

In essence, hist-PCA aptly addresses our unique challenges and limitations, enabling us to establish a streamlined pathway for efficient data sketching. This is particularly valuable in real-time, memory-constrained settings, ensuring both stability and accuracy, even when dealing with smaller batch inputs.

Detailed Description of Our *strAPI*

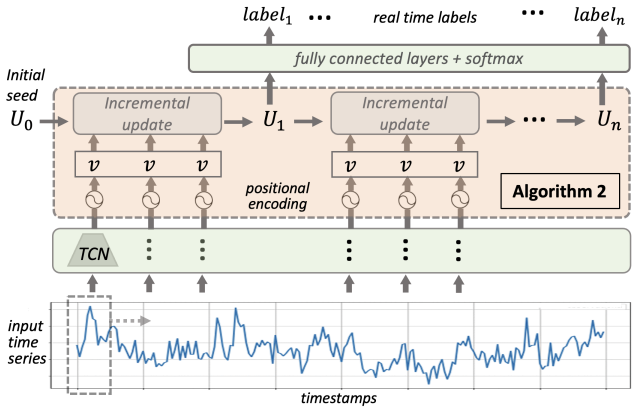


Figure 4: Our *strAPI* leverages TCN-acquired embeddings to update the time series sketch U using Algorithm 2 (highlighted in red). At anytime the current sketch U_i can be input to classification layer to produce the time series' class label.

Derived from hist-PCA (Algorithm 1), our proposed power algorithm is outlined in Algorithm 2, highlighting three key differentiating factors:

1. In line 2, positional encoding follows time series order, distinct from permutation-invariant samples in regular data streams. Positional information within time series is pivotal. We implement this using Continuous Augmented Positional Embeddings (CAPE) (Likhomanenko et al. 2021), known for computational efficiency and robustness in handling input length (Kalyan, Rajasekharan, and Sangeetha 2021; Zhai et al. 2023).
2. In line 3, the encoded batch undergoes transformation via fully-connected layers f_v, followed by relu activation. This introduces non-linearity, boosting the expressiveness of the learned sketch.
3. Lines 11 to 15 incorporate an attention model that assigns weights to new batches relative to the latest sketch. Line 15's update is a weighted summation of two elements: the first approximates the observed data covariance, while the second represents the new batch's covariance. These weights stem from the attention model in lines 11 to 13. Given the attention-driven nature of this power iteration, we dub Algorithm 2 as **Attentive Power Iteration (*API*)**.

Algorithm 2: our Attentive Power Iteration (*API*)

Input: a sequence of time series batch $\{X_1, ..., X_n\}$, batch size B, starting vector $U_0 \sim N(0, I_{m\times m})$, a positional encoding function *pos*
Output: final sketch U_n

Initialization : $U_1 \leftarrow U_0/\|U_0\|_2$
1: **for** $i = 1$ to n **do**
2: equip the input batch with their positional information by $\widetilde{X}_i \leftarrow X_i + pos(X_i)$
3: project the positional encoded batch to sketch space $V_i \leftarrow relu(f_v(\widetilde{X}_i))$
4: **if** $i = 1$ **then**
5: **while** U_1 not converge **do**
6: $U_1 \leftarrow U_1 + \frac{1}{B} V_1 (V_1^T U_1)$
7: $U_1 \leftarrow U_1/\|U_1\|_2$
8: **end while**
9: **else**
10: $U_i \leftarrow U_{i-1}$
11: get keys $K_i \leftarrow f_k([U_{i-1}(U_{i-1}^T U_i), \frac{1}{B} V_i (V_i^T U_i)])$
12: get query $Q_i \leftarrow f_q(U_{i-1}(U_{i-1}^T U_i))$
13: $H_i \leftarrow softmax(Q_i^T K_i)$
14: **while** U_i not converge **do**
15: $U_i \leftarrow H_{i,1} U_{i-1}(U_{i-1}^T U_i) + H_{i,2} \frac{1}{B} V_i (V_i^T U_i)$
16: $U_i \leftarrow U_i/\|U_i\|_2$
17: **end while**
18: **end if**
19: **end for**
20: **return** U_n

Extension : the 'while' loops can be run by κ iterations.

Having delineated the central aspect of our algorithm in Algorithm 2, we present the comprehensive framework of our *strAPI* in Figure 4. The highlighted red portion in Figure 4 signifies our Attentive Power Iteration (Algorithm 2). Besides, two supplementary modules are outlined as follows:

Temporal Convolutional Network (*TCN*). TCN (Oord et al. 2016) serves as our framework's initial component, amplifying the model's ability to grasp nonlinear patterns in

time series data. Shown in Figure 4, TCN employs a sliding window to extract nonlinear features from the time series. The ensuing sequence of TCN-derived embeddings is then funneled into Algorithm 2 to acquire their sketch [1].

Classification layer. At each time step during time series transfer, the latest sketch U_i can be input to a fully connected layer, followed by a softmax function, to predict a class probability distribution. Notably, the normalization steps in lines 7 and 16 of Algorithm 2 offer two benefits: 1) preventing sketch magnitude from becoming excessively large and enhancing convergence stability; 2) ensuring classification robustness to time series length. Without normalization, longer time series would yield significantly larger magnitudes of embeddings, causing disparities between time series of different lengths even within the same class.

Experiments

Our experiments aim to: a) evaluate *strAPI*'s performance and its accuracy relative to various baseline models across diverse model sizes, and b) assess the enhanced training and inference efficiency (including energy consumption, running time, and computational complexity) of *strAPI* compared to the baseline methods.

Experiment Setup

Datasets. Our evaluation encompasses five public benchmark datasets (Table 1). The Epilepsy dataset (Bagnall et al. 2018) simulates brain disorder recognition tasks among healthy participants. Handwriting (Shokoohi et al. 2017) captures smartwatch motion during alphabet writing. HAR-6 (Karim et al. 2019) employs smartphone sensors for human activity recognition. DSA-19 (Altun et al. 2010) involves data on 19 daily and sports activities. Google-13 (Warden 2018) is an audio dataset for keyword spotting systems. For each dataset, we perform 20 runs, randomly partitioning the dataset into training, validation, and test sets following proportions outlined in Table 1.

Baselines. We compare our *strAPI* against three deep-learning baselines for streaming time series classification:

1. *LSTM* (Hochreiter and Schmidhuber 1997): A Long Short-Term Memory network, a widely-used recurrent neural network (RNN) that excels at learning order dependence in sequence prediction tasks.
2. *SRNN* (Dennis et al. 2019): A Shallow-RNN, an advanced RNN variant tailored for resource-constrained edge devices within strict time and resource limits.
3. *TRNS* (Vaswani et al. 2017): We implement a transformer that gathers a sequence of SRNN embeddings over a specific timeframe and employs a temporal self-attentive module to weigh various sequence parts, facilitating holistic time series recognition.

Detailed model structures are provided in the last section.

Metrics. We gauge classification with *accuracy* (the fraction of correctly classified samples), ranging from 0 to 1 (achieving 1 for perfect matches). We also analyze *energy consumption* (kWh) and *running time* (seconds) for model cost, measured separately for training and inference. Moreover, we quantify *computational complexity* using Multiply-Accumulate Computations (MACs), a metric common in assessing computational load of deep learning models (Liang and Zou 2021; Lavin 2015). Our results reveal significantly reduced computational costs compared to the baselines, necessitating smaller memory buffers for storing intermediate results during computation.

Performance with Different Size Models

In Figure 5, we present a comparison of classification accuracy among our *strAPI* and three baselines. We specifically investigate their performance across varying model sizes based on parameter count. To achieve this, we implement three versions of each algorithm, approximately with $10,000$ (small), $15,000$ (medium), and $20,000$ (large) parameters. Detailed structures will be outlined in the last section. Across each dataset, we conduct 20 runs per algorithm per model size, recording mean accuracy and standard deviations. Our observations are as follows:

- In general, our *strAPI* demonstrates comparable accuracy with the three baselines on Google-13, DSA-19, and HAR-6 datasets. However, it notably achieves higher accuracy on the Epilepsy dataset (10+$\%$ improvement) and Handwriting dataset (45+$\%$ improvement). Our *strAPI* also exhibits greater stability compared to the three baselines, with an average standard deviation 15% smaller.
- We observe that the performance gap between our *strAPI* and the baselines widens as the length of the time series (#timestamps) increases. Specifically, our accuracy closely matches that of LSTM when #timestamps is around 100. However, when the time series length increases by 50+$\%$, our *strAPI* achieves 10+$\%$ higher accuracy, and this gap increases to 45+$\%$ when the length doubles. The poorer performance of the baselines might be attributed to the *catastrophic forgetting* effect observed in recurrent neural networks (RNNs) (Sodhani, Chandar, and Bengio 2020). This phenomenon characterizes the rapid forgetting of previously learned information by neural networks upon exposure to new data. In our memory-efficient model setup, the constrained representational capacity is further exacerbated by the use of fewer parameters. Furthermore, the sensitivity of RNNs to even minor weight changes becomes more pronounced over numerous iterations in long time series (Schak and Gepperth 2019). In contrast, our *strAPI* takes a different approach. It doesn't rely on a recurrent structure; instead, it employs attentive power iteration to create a linear weighted summation of local nonlinear representations. This approach effectively accentuates essential signature components with higher weights, leading to reduced susceptibility to the catastrophic forgetting effect.

Running Cost with Different Size Models

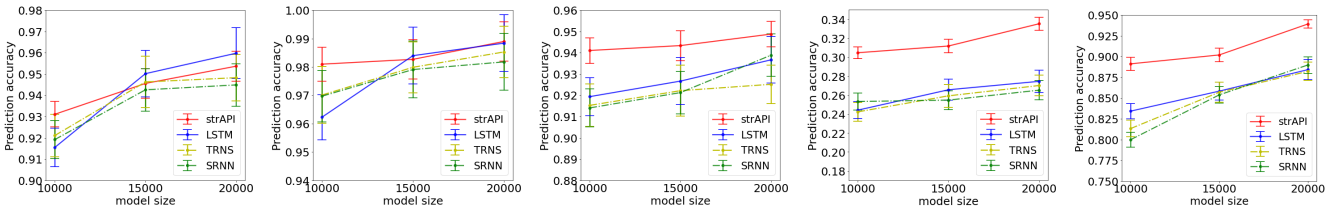
We employed *CodeCarbon*[2] to quantify energy consumption during training and inference across various algorithms.

[1] Thus, Algorithm 2 now utilizes the TCN-derived embedding sequence, rather than direct input from observed time series.

[2] https://github.com/mlco2/codecarbon

Dataset	source	# timestamps	# variables	# train	val	# test	# classes	description
Google-13	(Warden 2018)	99	32	51088	6798	6835	13	audio detection
DSA-19	(Altun et al. 2010)	125	45	3860	700	4560	19	sports activities
HAR-6	(Karim et al. 2019)	128	9	7352	1131	2947	6	daily activity
Handwriting	(Shokoohi et al. 2017)	152	3	150	23	850	26	writing motion
Epilepsy	(Bagnall et al. 2018)	206	3	137	21	138	4	bio-medical

Table 1: Dataset statistics and description.



(a) Google-13 (b) DSA-19 (c) HAR-6 (d) Handwriting (e) Epilepsy

Figure 5: Comparison of classification accuracy with different model sizes.

This open-source Python library provides energy consumption estimates by monitoring code execution. Experiments were conducted on a local machine equipped with 32GB of memory and an Intel Core i9 processor running at 2.9 GHz, without GPU support. Energy consumption estimates were derived by measuring power usage of both CPUs and RAM.

The energy consumption during training and testing is depicted in Figures 6(a) and 6(b). Remarkably, our *strAPI* achieved over **70% reduction in energy consumption compared to LSTM, and 30% (48%) less energy during training (inference) when contrasted with SRNN on average**. Notably, *strAPI*'s energy efficiency increases with larger model sizes. Figures 6(c) and 6(d) present comparisons of running time during training and testing in log scale. Our *strAPI* outperforms LSTM by **threefold in terms of speed**, and about 1.3 times faster than SRNN.

Additionally, we measured Multiply-Accumulate Computations (MACs) in Figure 6(e), finding that *strAPI*'s MACs are only about one-fifth of LSTM's and one-third of SRNN's. This complexity comparison elucidates the greater efficiency of our *strAPI* in terms of energy and time when compared to RNNs. While RNNs involve transformations across both feedforward and recurrent dimensions (Zhang et al. 2016), our *strAPI* computations are confined within a single time series batch during each update.

Experiments on Streaming Setting

We conducted streaming scenario tests on Epilepsy dataset, known for its long time series. Our approach involves offline model training followed by testing in a streaming setting, where time series samples arrive as continuous streams ordered by timestamp. Figure 7 displays accuracy across time, revealing the following key observations: 1) Initially, our *strAPI* exhibits lower accuracy on the first batch. This discrepancy could arise from the initial variance in covariance estimation. 2) Notably, our *strAPI* rapidly achieves high performance. By the time the second batch arrives, it attains accuracy exceeding 0.87, even though the majority of the time series remains unobserved. This early accuracy level closely approximates the final performance of other baselines when they have access to the complete time series. 3) Our *strAPI* achieves its peak performance upon the arrival of the third and fourth batches. Subsequently, it sustains high accuracy throughout the entire time series. This experiment underscores our *strAPI*'s efficacy, efficiency, and stability in the context of streaming scenarios for time series classification.

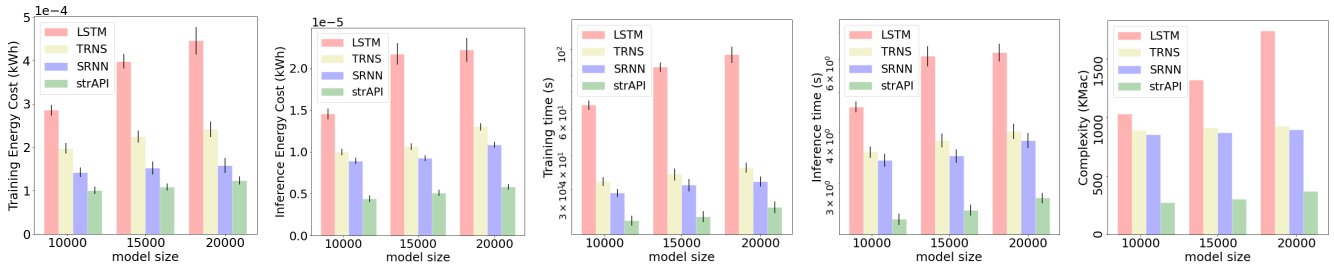
Ablation Study

We conducted an ablation study on our *strAPI* framework to analyze the contributions of its different components. The study was carried out on the HAR-6 dataset. Figure 8 depicts the accuracy achieved by five versions of *strAPI*: one full version and four ablated versions, each with a specific part removed. When ablating a component, we ensured that the total number of parameters in the model remained constant.

In summary, the full model outperformed the ablated versions, highlighting the synergistic integration of all components. Notably, the version lacking the Temporal Convolutional Network (TCN) exhibited the lowest performance, underscoring TCN's pivotal role in learning nonlinear local features from time series data. The second poorest performance arose from replacing Algorithm 2 with Algorithm 1 (i.e., no attentive weighting on data streams). This result underscores the efficacy of attentive weighting in capturing temporal patterns. Moreover, the importance of position encoding (line 2 in Algorithm 2) and embedding projection (line 3 in Algorithm 2) is evident in their contribution to achieving higher classification accuracy.

Implementation Details and Sensitivity Study

All experiments were implemented using the *PyTorch* framework. We employed a training batch size of 100 and utilized the *Adam* optimizer with a learning rate of 1×10^{-3} for training our model. The parameter settings for our *strAPI*



(a) Training energy (b) Inference energy (c) Training time (d) Inference time (e) Complexity

Figure 6: Comparison of running costs and complexity with different model sizes.

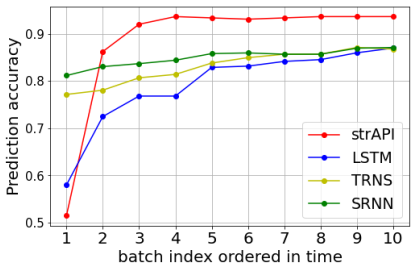


Figure 7: Classification performance in streaming scenario.

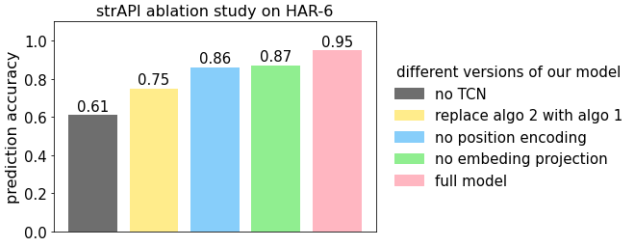


Figure 8: Ablation study of our model.

framework are as follows: 1) TCN consists of two layers. Both layers have a kernel size of 3 and output dimension of 30. 2) In Algorithm 2, the critical hyperparameters include the sketch dimensions p and attention space dimensions q. The sketch has a shape of $R^{m\times k}$, where m is the sketch dimensions and k is the number of sketches. To ensure ample representation capacity while maintaining a compact model size, we set $k = 2$. 3) Each batch contains 10 samples (embeddings) by default. 4) We employ $\kappa = 1$ iteration (while loop in Algorithm 2) for its effectiveness and high efficiency.

We constructed small, medium, and large *strAPI* models by varying the sketch size p and attention dimensions q. Specifically, the small model has $p = 20$ and $q = 20$, the medium model has $p = 25$ and $q = 25$, while the large model has $p = 30$ and $q = 40$.

For the three baseline models, we utilized two LSTM layers to ensure high efficiency in both memory usage and running time, following the settings used in (Dennis et al. 2019). More specifically, the small model had first and second output dimensions of $[32, 10]$, the medium model had $[32, 20]$, and the large model had $[32, 32]$. The attention dimensions in the TRNS model were set to 30.

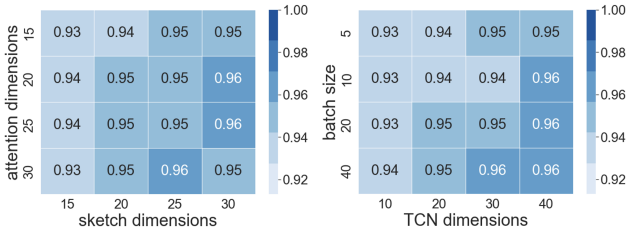


Figure 9: Parameter sensitivity analysis on HAR-6 dataset.

Figure 9 illustrates the robustness of our *strAPI* method across four key hyperparameters. Impressively, our method consistently demonstrates high and stable accuracy. It is important to consider that: 1) In practice, it is advisable to set sketch dimensions and TCN dimensions to be sufficiently large for capturing comprehensive and informative patterns. 2) Batch size often presents a trade-off: smaller batches facilitate timely analysis without delay, while larger batches enable more efficient computation. Our *strAPI* exhibits robustness to both small (5) and large (40) batch sizes.

Conclusion

In summary, we introduce *strAPI*, a pioneering streaming neural network that incorporates incremental weighted PCA within a supervised latent space, augmented by temporal attention weights for individual samples. This innovative approach yields real-time principal directions that adeptly encapsulate the core essence of time series data, facilitating swift and precise classification tasks on resource-constrained environment. Remarkably, *strAPI* excels in streaming scenarios for time series classification, even without requiring a complete observation of the entire time series. Experimental results underscore its exceptional performance, showcasing not only superior accuracy but also significantly reduced energy consumption, enhanced running speed, and diminished computational complexity. Importantly, our focus is on classifying entire time series into a single class, thereby mitigating concerns related to concept drift. Moving forward, we plan to explore *strAPI*'s potential in dynamic pattern recognition scenarios characterized by concept drift.

Acknowledgments

This work was supported in part by United State, Department of Energy, under Contract Number DE-SC0012704.

References

Altun, K.; Barshan, B.; Barshan, B.; and Tunçel, O. 2010. Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition*, 43(10): 3605–3620.

Bagnall, A.; Dau, H. A.; Lines, J.; Flynn, M.; Large, J.; Bostrom, A.; Southam, P.; and Keogh, E. 2018. The UEA multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166.

Bloomfield, P. 2004. *Fourier analysis of time series: an introduction*. John Wiley & Sons.

Cardot, H.; and Degras, D. 2018. Online principal component analysis in high dimension: Which algorithm to choose? *International Statistical Review*, 86(1): 29–50.

Chen, B.; Yu, S.; Yu, Y.; and Zhou, Y. 2020. Acoustical damage detection of wind turbine blade using the improved incremental support vector data description. *Renewable Energy*, 156: 548–557.

Chen, X.; Jayaram, R.; Levi, A.; and Waingarten, E. 2022. New streaming algorithms for high dimensional EMD and MST. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, 222–233.

Coleman, B.; and Shrivastava, A. 2020. Sub-linear race sketches for approximate kernel density estimation on streaming data. In *Proceedings of The Web Conference 2020*, 1739–1749.

Dennis, D. K.; Acar, D.; Mandikal, V.; Sadasivan, V.; Simhadri, H.; Saligrama, V.; and Jain, P. 2019. Shallow RNNs: A method for accurate time-series classification on tiny devices. In *Proc. 33rd Conf. Neural Inf. Process. Syst.(NeurIPS)*, 8–14.

Diakonikolas, I.; Kane, D. M.; Pensia, A.; and Pittas, T. 2022. Streaming algorithms for high-dimensional robust statistics. In *International Conference on Machine Learning*, 5061–5117. PMLR.

Falcone, R.; Anderlucci, L.; and Montanari, A. 2022. Matrix sketching for supervised classification with imbalanced classes. *Data Mining and Knowledge Discovery*, 36(1): 174–208.

Fujiwara, T.; Chou, J.-K.; Shilpika, S.; Xu, P.; Ren, L.; and Ma, K.-L. 2019. An incremental dimensionality reduction method for visualizing streaming multidimensional data. *IEEE transactions on visualization and computer graphics*, 26(1): 418–428.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Huang, H.; and Kasiviswanathan, S. P. 2015. Streaming anomaly detection using randomized matrix sketching. *Proceedings of the VLDB Endowment*, 9(3): 192–203.

Jeong, Y.-S.; Jeong, M. K.; and Omitaomu, O. A. 2011. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9): 2231–2240.

Jin, T.; Huang, K.; Tang, J.; and Xiao, X. 2021. Optimal streaming algorithms for multi-armed bandits. In *International Conference on Machine Learning*, 5045–5054. PMLR.

Kalyan, K. S.; Rajasekharan, A.; and Sangeetha, S. 2021. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*.

Karim, F.; Majumdar, S.; Darabi, H.; and Harford, S. 2019. Multivariate LSTM-FCNs for time series classification. *Neural Networks*.

Kowshik, S.; Nagaraj, D.; Jain, P.; and Netrapalli, P. 2021. Near-optimal offline and streaming algorithms for learning non-linear dynamical systems. *Advances in Neural Information Processing Systems*, 34: 8518–8531.

Lavin, A. 2015. maxdnn: An efficient convolution kernel for deep learning with maxwell gpus. *arXiv preprint arXiv:1501.06633*.

Liang, W.; and Zou, J. 2021. Neural group testing to accelerate deep learning. In *2021 IEEE International Symposium on Information Theory (ISIT)*, 958–963. IEEE.

Liberty, E. 2013. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 581–588.

Likhomanenko, T.; Xu, Q.; Synnaeve, G.; Collobert, R.; and Rogozhnikov, A. 2021. Cape: Encoding relative positions with continuous augmented positional embeddings. *Advances in Neural Information Processing Systems*, 34: 16079–16092.

Lin, F.; and Cohen, W. W. 2010. Power iteration clustering.

Makarychev, Y.; Manoj, N. S.; and Ovsiankin, M. 2022. Streaming algorithms for ellipsoidal approximation of convex polytopes. In *Conference on Learning Theory*, 3070–3093. PMLR.

Mitliagkas, I.; Caramanis, C.; Jain, P.; and Caramanis, C. 2013. Memory limited, streaming PCA. *Advances in neural information processing systems*, 26.

Mohammadi, M.; Al-Fuqaha, A.; Sorour, S.; and Guizani, M. 2018. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4): 2923–2960.

Oja, E.; and Karhunen, J. 1985. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1): 69–84.

Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Ribeiro, A. H.; Tiels, K.; Aguirre, L. A.; and Schön, T. 2020. Beyond exploding and vanishing gradients: analysing RNN training using attractors and smoothness. In *International*

conference on artificial intelligence and statistics, 2370–2380. PMLR.

Ross, D. A.; Lim, J.; Lin, R.-S.; and Yang, M.-H. 2008. Incremental learning for robust visual tracking. *International journal of computer vision*, 77: 125–141.

Salehinejad, H.; Sankar, S.; Barfett, J.; Colak, E.; and Valaee, S. 2017. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*.

Schak, M.; and Gepperth, A. 2019. A study on catastrophic forgetting in deep LSTM networks. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Deep Learning: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part II 28*, 714–728. Springer.

Shokoohi, M.; Hu, B.; Jin, H.; Wang, J.; and Keogh, E. 2017. Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data mining and knowledge discovery*, 31: 1–31.

Sodhani, S.; Chandar, S.; and Bengio, Y. 2020. Toward training recurrent neural networks for lifelong learning. *Neural computation*, 32(1): 1–35.

Song, Y.; Sebe, N.; Wang, W.; and Sebe, N. 2022a. Improving covariance conditioning of the SVD meta-layer by orthogonality. In *European Conference on Computer Vision*, 356–372. Springer.

Song, Y.; Sebe, N.; Wang, W.; and Sebe, N. 2022b. Orthogonal SVD Covariance Conditioning and Latent Disentanglement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Tyralis, H.; and Papacharalampous, G. 2017. Variable selection in time series forecasting using random forests. *Algorithms*, 10(4): 114.

Van Houdt, G.; Mosquera, C.; and Nápoles, G. 2020. A review on the long short-term memory model. *Artificial Intelligence Review*, 53: 5929–5955.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Warden, P. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*.

Yang, P.; Hsieh, C.-J.; Wang, J.-L.; and Hsieh, C.-J. 2018. History PCA: A new algorithm for streaming PCA. *arXiv preprint arXiv:1802.05447*.

Yi, Q.; Wang, C.; Wang, K.; and Wang, Y. 2022. Effective streaming low-tubal-rank tensor approximation via frequent directions. *IEEE Transactions on Neural Networks and Learning Systems*.

Yoo, S.; Huang, H.; and Kasiviswanathan, S. P. 2016. Streaming spectral clustering. In *2016 IEEE 32nd international conference on data engineering (ICDE)*, 637–648. IEEE.

Zhai, S.; Likhomanenko, T.; Littwin, E.; Busbridge, D.; Ramapuram, J.; Zhang, Y.; Gu, J.; and Susskind, J. M. 2023. Stabilizing Transformer Training by Preventing Attention Entropy Collapse. In *International Conference on Machine Learning*, 40770–40803. PMLR.

Zhang, S.; Wu, Y.; Che, T.; Lin, Z.; Memisevic, R.; Salakhutdinov, R. R.; and Bengio, Y. 2016. Architectural complexity measures of recurrent neural networks. *Advances in neural information processing systems*, 29.

Zhang, Z.; Ng, I.; Gong, D.; Liu, Y.; Abbasnejad, E.; Gong, M.; Zhang, K.; and Shi, J. Q. 2022. Truncated matrix power iteration for differentiable DAG learning. *Advances in Neural Information Processing Systems*, 35: 18390–18402.