LETS-C: Leveraging Text Embedding for Time Series Classification

Rachneet Kaur **Zhen Zeng** **Tucker Balch** **Manuela Veloso**
J.P. Morgan AI Research
{rachneet.kaur, zhen.zeng, tucker.balch, manuela.veloso}@jpmorgan.com

Abstract

Recent advancements in language modeling have shown promising results in time series data analysis, with fine-tuning pre-trained large language models (LLMs) achieving state-of-the-art (SOTA) performance on standard benchmarks. However, LLMs-based methods require millions of trainable parameters, presenting a significant drawback due to their large size. We propose an alternative approach to leveraging the success of language modeling in the time series domain. Instead of fine-tuning LLMs, we utilize a text embedding model to embed time series and then pair the embeddings with a simple classification head composed of convolutional neural networks and multilayer perceptron. We conducted extensive experiments on a well-established time series classification benchmark. We demonstrated LETS-C not only outperforms the current SOTA in classification accuracy but also offers a lightweight solution, using only 14.5% of the trainable parameters compared to the SOTA model. Our findings suggest that leveraging text embedding models to encode time series data, combined with a simple yet effective classification head, offers a promising direction for achieving high-performance time series classification while maintaining a relatively lightweight model architecture.

1 Introduction

Time series classification Bagnall et al. (2017); Abanda et al. (2019); Ismail Fawaz et al. (2019) has gained significant attention due to its applications across domains, such as finance Passalis et al. (2017), healthcare Lipton et al. (2016), and activity recognition Yang et al. (2015). The increasing availability of time series data has driven the need for efficient and accurate classification methods. Advances in natural language processing (NLP) and large language models (LLMs) have shown promises in language modeling Achiam et al. (2023), particularly in capturing temporal dependencies within sequential data. Inspired by this success, researchers have extended these techniques to time series analysis by prompting Gruver et al. (2023) and fine-tuning pre-trained LLMs Zhou et al. (2024); Jin et al. (2023), achieving state-of-the-art (SOTA) performance on established benchmarks.

However, using LLMs for time series classification has drawbacks due to their large size, making them computationally expensive and unsuitable for resource-limited settings Bommasani et al. (2021). At training time, fine-tuning partially frozen pre-trained LLMs also often involves millions of trainable parameters Zhou et al. (2024). To address this, we propose an alternative approach to leverage the success of language modeling in the time series domain. In particular, we propose LETS-C (**L**everaging Text **E**mbeddings for **T**ime **S**eries **C**lassification), which utilizes off-the-shelf text embedding models instead of fine-tuning LLMs for time series classification. To the best of our knowledge, this work is the **first** to explore the potential of text embeddings in time series analysis, specifically classification, and demonstrate SOTA performance. LETS-C combines text embeddings with a simple yet effective classification head composed of convolutional neural networks (CNNs) and a multilayer perceptron (MLP). By projecting time series data using text embedding models, we capture the intricate patterns and dependencies present in the temporal data. The embeddings and time series are then fed into the classification head, which learns to discriminate between different

NeurIPS 2024 Workshop on Time Series in the Age of Large Models.

classes. Through extensive testing on a comprehensive benchmark with various time series datasets, we showed that LETS-C outperforms 20 baselines, including the previous SOTA, and is significantly more efficient, using far fewer trainable parameters.

Our main contributions are: **1) Text Embeddings for Time Series:** We introduce LETS-C, the **first** work to leverage text embeddings for time series analysis, specifically for classification; **2) State-of-the-Art Performance:** LETS-C achieves SOTA classification accuracy on a well-established benchmark with diverse time series datasets, surpassing 20 baselines; **3) Lightweight:** LETS-C is significantly more efficient, achieving higher accuracy with much fewer trainable parameters (14.5%) compared to the existing SOTA. Additionally, we conducted comprehensive analyses to showcase LETS-C's effectiveness: **1) Text Embeddings Models:** LETS-C, using various text embedding models, consistently outperforms previous SOTA with fewer trainable parameters, validating our approach's generalizability; **2) Time Series Embeddings:** We showed that text embeddings enhance time series classification by demonstrating that embeddings from the same class are more similar than those from different classes, thereby boosting accuracy; **3) Trade-off between Accuracy and Model Size:** LETS-C maintains high accuracy with considerably smaller model sizes, enhancing computational efficiency without significantly compromising accuracy. See Appendix Section A for related work on time series classification, language model applications, and text embeddings.

2 Methodology

Given a time series classification dataset $\mathcal{D} = (\mathbf{x_i}, y_i)_{i=1}^{N}$, where $\mathbf{x_i}$ is a multivariate time series and $y_i \in 1, 2, \ldots, C$ is the class label, the objective is to train a classifier that accurately predicts the class label $\hat{y}_i$. As shown in Figure 1, we introduce the LETS-C framework that leverages text embeddings for classifying time series.

Specifically, we: 1) min-max normalize $\mathbf{x_i}$ into $\bar{\mathbf{x}}_\mathbf{i}$, 2) generate text embeddings of $\bar{\mathbf{x}}_\mathbf{i}$, 3) fuse these embeddings with $\bar{\mathbf{x}}_\mathbf{i}$, and finally 4) feed the fused representation to a classification head comprising CNNs and MLP. This simple classifier aims to test the hypothesis that text embeddings provide strong representations for effective classification.

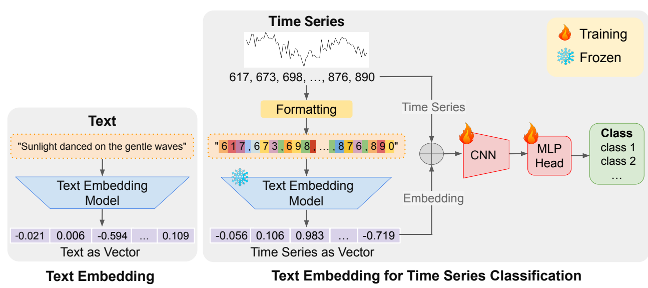

Figure 1: **Left:** Conventional text embedding. **Right:** LETS-C

Preprocessing Each dimension of time series $\mathbf{x_i}$ is min-max normalized to $[0, 1]$ using training data.

Text Embedding of Time Series It is crucial to carefully format the preprocessed time series into strings before using text embeddings, as the tokenization of numerical strings can significantly affect the embeddings. Liu & Low (2023) has shown that tokenization impacts a model's arithmetic abilities, with commonly used subword tokenization methods like Byte Pair Encoding (BPE) arbitrarily subdividing numbers, causing similar numbers to appear very differently. To mitigate this, we adopted a digit-space tokenization strategy, spacing each digit and omitting decimal points for precision Gruver et al. (2024). For example, the series 0.645, 6.45, 64.5, 645.0 would be tokenized as "6 4 , 6 4 5 , 6 4 5 0 , 6 4 5 0 0" with two-decimal precision. This method ensures separate tokenization of each digit, preserving numerical integrity and enhancing pattern recognition in language models. Next, we used the `text-embedding-3-large` model OpenAI (2024) to embed the formatted time series. We transformed each dimension of $\mathbf{x_i} \in \mathbb{R}^{d \times l_x}$ into text embeddings $\mathbf{e_i} \in \mathbb{R}^{d \times l_e}$, where d represents the multivariate dimension, and l_x and l_e are the lengths of the time series and embeddings, respectively. Note that embedding computation happens only once; the embeddings can be stored and reused, in contrast to ongoing costs associated with fine-tuning LLMs Zhou et al. (2024).

Fusing Embedding and Time Series The embeddings are fused with the time series data through addition, using zero padding to align lengths and combine the strengths of both modalities for enhanced model performance. Fusing embeddings from different modalities is well-supported in the literature Manzoor et al. (2023); Guo et al. (2019); Poria et al. (2018), with various approaches, including element-wise addition, successfully integrating embeddings from multiple modalities.

Lightweight Classification Head Lastly, we pair the fused time series representation with a simple classification head composed of 1D CNNs and an MLP for time series classification. With a simple classification head, our model is lightweight and requires much less trainable parameters (Section 3.2).

3 Experiments

3.1 Experimental Setup

Datasets and Evaluation Metrics We evaluated LETS-C against a well-established benchmark from the UEA Archive (Bagnall et al., 2018), including 10 datasets curated for challenging and diverse domains. These datasets feature multivariate dimensions ranging from 3 to 963, lengths up to 1751, and up to 26 classes. Further details are provided in Appendix Section B. To assess the classifiers, we used metrics such as classification accuracy and *AvgWins*. *AvgWins* is defined as the average number of times a method outperforms others across datasets, counting ties. We also assessed the models' computational efficiency by examining trainable parameters and training/inference times.

Baselines We included 20 baselines for a thorough comparison: *Classical methods:* 1) Dynamic Time Warping (DTW), 2) eXtreme Gradient Boosting (XGBoost), and 3) RandOm Convolutional KErnel Transform (ROCKET); *MLP-based methods:* 4) LightTS, and 5) DLinear; *RNN-based models:* 6) Long Short-Term Memory (LSTM), 7) Long- and Short-term Time-series Network (LSTNet), and 8) Linear State Space Layer (LSSL); *CNN-based models:* 9) Temporal Convolutional Network (TCN), and 10) TimesNet; *Transformer-based models:* 11) Transformer, 12) Reformer, 13) Informer, 14) Pyraformer, 15) Autoformer, 16) Stationformer, 17) FEDformer, 18) ETSformer, 19) Flowformer; and *LLM-based model:* 20) OneFitsAll. For details on these methods, see Appendix Section C.

Implementation Details Experiments were run on a Linux machine with an NVIDIA T4 GPU and 16GB RAM, using PyTorch v2.4.0 on Python 3.11, with reproducibility ensured by a fixed random seed and default RAdam optimizer settings $(\beta_1, \beta_2) = (0.9, 0.999)$. We tested LETS-C with four text embedding models, including `text-embedding-3-large`, as detailed in Section 3.2. Exploratory hyperparameter optimization identified 1-3 1D convolutional and 1-2 linear layers as optimal for LETS-C across all datasets (see Appendix Section D for hyperparameter details). For tokenization, maintaining a one decimal precision optimized performance, as detailed in Appendix Section J.

3.2 Results

Comparison to State-of-the-art Table 1 shows a comparative analysis of our LETS-C approach versus 20 baselines[1]. LETS-C consistently demonstrates robust performance across all datasets, with the highest average accuracy (76.16%) and AvgWins (40%). This surpasses the recent SOTA, OneFitsAll (accuracy: 73.97%, AvgWins: 20%), and TimesNet (accuracy: 73.57%, AvgWins: 0%). Notably, LETS-C significantly outperforms OneFitsAll on 6 out of 10 datasets and excels on challenging datasets like PEMS-SF, with 963 features, and EthanolConcentration, with a time series length of 1751. These results establish LETS-C as a new benchmark for time series classification.

Table 1: Comparison of classification accuracy (%) and AvgWins (%). **Red:** Best, Blue: Second best.

Model/Dataset		EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↑**	**AvgWins % ↑**
Classical methods	DTW	32.3	52.9	28.6	71.7	94.9	71.1	77.7	53.9	96.3	90.3	66.97	0%
	XGBoost	43.7	63.3	15.8	73.2	86.5	98.3	84.6	48.9	69.6	75.9	65.98	10%
	ROCKET	45.2	64.7	58.8	75.6	96.2	75.1	90.8	53.3	71.2	94.4	72.53	20%
MLP	LightTS	29.7	67.5	26.1	75.1	96.2	88.4	89.8	51.1	100	80.3	70.42	10%
	DLinear	32.6	68	27	75.1	96.2	75.1	87.3	50.5	81.4	82.1	67.53	0%
RNN	LSTM	32.3	57.7	15.2	72.2	79.7	39.9	68.9	46.6	31.9	41.2	48.56	0%
	LSTNet	39.9	65.7	25.8	77.1	98.1	86.7	84	52.8	100	87.8	71.79	10%
	LSSL	31.1	66.7	24.6	72.7	98.4	86.1	90.8	52.2	100	85.9	70.85	10%
CNN	TCN	28.9	52.8	53.3	75.6	98.9	68.8	84.6	55.6	95.6	88.4	70.25	0%
	TimesNet	35.7	68.6	32.1	78	98.4	89.6	91.8	57.2	99	85.3	73.57	0%
Transformers	Transformer	32.7	67.3	32	76.1	98.7	82.1	92.2	53.9	98.4	85.6	71.9	0%
	Reformer	31.9	68.6	27.4	77.1	97.8	82.7	90.4	56.7	97	85.6	71.52	0%
	Informer	31.6	67	32.8	80.5	98.9	81.5	90.1	53.3	100	85.6	72.13	20%
	Pyraformer	30.8	65.7	29.4	75.6	98.4	83.2	88.1	53.3	99.6	83.4	70.75	0%
	Autoformer	31.6	68.4	36.7	74.6	96.2	82.7	84	50.6	100	85.9	71.07	10%
	Stationformer	32.7	68	31.6	73.7	99.2	87.3	89.4	57.2	100	87.5	72.66	20%
	FEDformer	31.2	66	28	73.7	98.4	80.9	88.7	54.4	100	85.3	70.66	10%
	ETSformer	28.1	66.3	32.5	71.2	95.9	86	89.6	55	100	85	70.96	10%
	Flowformer	33.8	67.6	33.8	77.6	98.9	83.8	92.5	56.1	98.8	86.6	72.95	0%
LLM	OneFitsAll	34.2	69.2	32.7	77.2	98.6	87.9	93.2	59.4	99.2	88.1	73.97	20%
	LETS-C	52.9	68.9	23.8	78	99.2	93.1	93.2	62.8	99.2	90.6	**76.17**	**40%**

Computational Cost Analysis Table 2 provides a detailed analysis of the trainable parameters associated with LETS-C compared to the previous SOTA model, OneFitsAll. LETS-C achieved higher performance with only 14.48% of the trainable parameters on average, compared to OneFitsAll. Further, LETS-C is more computationally efficient in training and inference than OneFitsAll, with detailed time comparisons in Appendix Section F.

[1] To maintain consistency with prior works, we use the same settings as in TimesNet and OneFitsAll, where the UEA benchmark includes only training and testing sets (with no validation set). Consequently, the reported performance reflects the model's upper bound measurement.

Table 2: Comparison of trainable parameters (millions) for LETS-C vs. OneFitsAll, with the Ratio (%) $= 100 \times \frac{\text{\# of LETS-C parameters}}{\text{\# of OneFitsAll parameters}}$ illustrating LETS-C's efficiency relative to OneFitsAll.

Model/Dataset	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	Average ↓
LETS-C (Ours)	0.28	0.003	0.15	0.04	0.14	0.56	0.30	0.33	0.14	0.26	0.22
OneFitsAll	1.42	2.37	1.73	2.03	1.32	10.23	0.98	1.04	1.82	1.0	2.39
Ratio (%)	19.89	0.16	8.89	2.28	11.19	5.51	30.83	32.06	7.77	26.21	**14.48**

Ablation Study To assess the benefits of fusing text embeddings with time series data versus using them separately, we conducted an ablation study. Results in Appendix Section G and Table 6 show that combining both embeddings and time series data achieves the highest average accuracy (76.17%) and the most AvgWins, with accuracy dropping to 72.94% without the time series and 73.47% without the embeddings, highlighting the significant gains from fusion for optimal accuracy.

Alternative Methods for Fusing Time Series with Embeddings We explored two additional fusion methods for time series and embeddings: a *Fusion network* processing data through separate layers before merging, and *Concatenation*, where data are concatenated and processed through a lightweight head. Appendix Section H and Table 7 detail the performance of these methods. The simple addition method in LETS-C yielded the highest accuracy (76.11%, compared to 73.40% for Fusion and 74.22% for Concatenation), with both alternative methods increasing parameter count and complexity.

Power of Text Embeddings on Time Series To assess text embeddings' effectiveness for time series, we analyzed average cosine similarities within and between classes, and averaged these similarities across channels for multivariate series. This evaluated closeness in the embedding space, both intra-class and inter-class. Figure 2 (left) shows these similarities in heatmaps, scaled from 0 to 1 via min-max normalization, with warmer colors indicating higher similarities. The heatmaps confirm that intra-class similarities consistently surpass inter-class ones, proving text embeddings effectively retain and convey significant information from the underlying time series data.

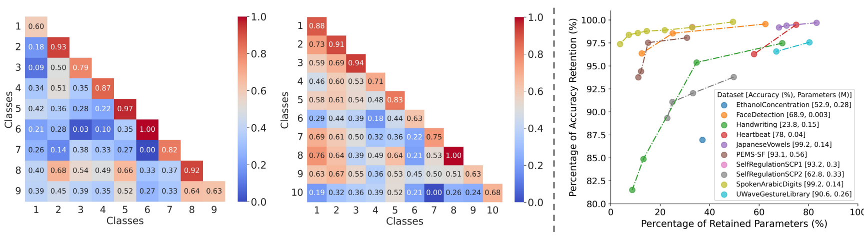

Figure 2: **Left:** Heatmaps show within- and between-class cosine similarities from JapaneseVowels **(far left)**, with 9 classes, and SpokenArabicDigits **(middle)**, with 10 classes. **Right:** Trade-off between accuracy and parameter retention percentages relative to LETS-C's optimal values.

Text Embedding Models To test LETS-C's generalization across different text embeddings, we evaluated it using 3 additional models: `e5-mistral-7b-instruct` Wang et al. (2023), `gte-large-en-v1.5` Li et al. (2023), and `nomic-embed-text-v1` Nussbaum et al. (2024). These models differ in embedding dimensions, token lengths, and sizes (Appendix K, and Table 10). LETS-C consistently surpasses SOTA in accuracy, using far fewer trainable parameters across various embedding models. Specifically, `text-embedding-3-large` achieves 76.17% accuracy, while `e5-mistral-7b-instruct`, `gte-large-en-v1.5`, and `nomic-embed-text-v1` achieve 74.89%, 76.02%, and 75.12% resp. These results exceed the SOTA accuracy of 73.97% and require significantly fewer parameters—14.48%, 15.62%, 9.24%, and 5.31% of OneFitsAll's parameter count.

Trade-offs: Model Accuracy vs. Parameter Complexity Next, we aim to explore the trade-offs between model accuracy and size. Figure 2 (right) shows the trade-off between accuracy retention and parameter retention across all datasets, compared to the optimal metrics of our LETS-C model (see Tables 1 and 2). The figure demonstrates LETS-C's ability to maintain high accuracy with fewer parameters across datasets (see Appendix Table 11 for details). We analyzed this trade-off in specific datasets-Heartbeat, PEMS-SF, and SpokenArabicDigits (Appendix Section L), showing that LETS-C, with fewer parameters, maintains sufficient accuracy for resource-limited applications.

4 Conclusion

We introduced LETS-C, a novel approach that utilizes text embeddings for time series classification, marking its first use in this field. By mapping time series data through text embedding models and employing a simple classification head, LETS-C achieves SOTA performance on a comprehensive benchmark, using fewer trainable parameters. Our analysis highlights LETS-C's effectiveness with various text embedding models, and the trade-offs between model accuracy and size.

References

Amaia Abanda, Usue Mori, and Jose A Lozano. A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2):378–412, 2019.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, 31:606–660, 2017.

Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

Claus Bahlmann, Bernard Haasdonk, and Hans Burkhardt. Online handwriting recognition with support vector machines-a kernel approach. In *Proceedings eighth international workshop on frontiers in handwriting recognition*, pp. 49–54. IEEE, 2002.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Mouldi Bedda and Nacereddine Hammami. Spoken Arabic Digit. UCI Machine Learning Repository, 2010. DOI: https://doi.org/10.24432/C52C9Q.

Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pp. 359–370, 1994.

Niels Birbaumer, Nimr Ghanayim, Thilo Hinterberger, Iver Iversen, Boris Kotchoubey, Andrea Kübler, Juri Perelmouter, Edward Taub, and Herta Flor. A spelling device for the paralysed. *Nature*, 398(6725):297–298, 1999.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Defu Cao, Furong Jia, Sercan O Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. Tempo: Prompt-based generative pre-trained transformer for time series forecasting. *arXiv preprint arXiv:2310.04948*, 2023.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Marco Cuturi. Fast global alignment kernels. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 929–936, 2011.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.

Jonas Geiping and Tom Goldstein. Cramming: Training a language model on a single gpu in one day. In *International Conference on Machine Learning*, pp. 11117–11143. PMLR, 2023.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters, 2023.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36, 2024.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Wenzhong Guo, Jianwen Wang, and Shiping Wang. Deep multimodal representation learning: A survey. *Ieee Access*, 7:63373–63394, 2019.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

Young-Seon Jeong, Myong K Jeong, and Olufemi A Omitaomu. Weighted dynamic time warping for time series classification. *Pattern recognition*, 44(9):2231–2240, 2011.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Yushan Jiang, Zijie Pan, Xikun Zhang, Sahil Garg, Anderson Schneider, Yuriy Nevmyvaka, and Dongjin Song. Empowering time series analysis with large language models: A survey. *ArXiv*, abs/2402.03182, 2024. URL `https://api.semanticscholar.org/CorpusID:267412144`.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*, 2023.

Argyro Kampouraki, George Manis, and Christophoros Nikou. Heartbeat time series classification with support vector machines. *IEEE transactions on information technology in biomedicine*, 13(4): 512–518, 2008.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Mineichi Kudo, Jun Toyama, and Masaru Shimbo. Japanese Vowels. UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C5NS47.

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

James Large, E Kate Kemsley, Nikolaus Wellner, Ian Goodall, and Anthony Bagnall. Detecting forged alcohol non-invasively through vibrational spectroscopy and machine learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 298–309. Springer, 2018.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models. *arXiv preprint arXiv:2402.14848*, 2024.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

Zachary C Lipton, David Kale, and Randall Wetzel. Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. In *Machine learning for healthcare conference*, pp. 253–270. PMLR, 2016.

Chengyu Liu, David Springer, Qiao Li, Benjamin Moody, Ricardo Abad Juan, Francisco J Chorro, Francisco Castells, José Millet Roig, Ikaro Silva, Alistair EW Johnson, et al. An open access database for the evaluation of heart sound algorithms. *Physiological measurement*, 37(12):2181, 2016.

Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6): 657–675, 2009.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019a.

Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*, 2021.

Tiedong Liu and Bryan Kian Hsiang Low. Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks. *arXiv preprint arXiv:2305.14201*, 2023.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019b.

Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. *Advances in Neural Information Processing Systems*, 35: 9881–9893, 2022.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2421–2425, 2024.

Muhammad Arslan Manzoor, Sarah Albarri, Ziting Xian, Zaiqiao Meng, Preslav Nakov, and Shangsong Liang. Multimodality representation learning: A survey on evolution, pretraining and its applications. *ACM Transactions on Multimedia Computing, Communications and Applications*, 20 (3):1–34, 2023.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder, 2024.

OpenAI. OpenAI Embedding Models. `https://platform.openai.com/docs/guides/embeddings`, `https://openai.com/index/new-embedding-models-and-api-updates/`, 2024. Accessed: 2024-05-13.

Victor Pan. Fast approximate computations with cauchy matrices and polynomials. *Mathematics of Computation*, 86(308):2799–2826, 2017.

Victor Y Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2012.

Nikolaos Passalis, Avraam Tsantekidis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Time-series classification using neural bag-of-features. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pp. 301–305. IEEE, 2017.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Soujanya Poria, Navonil Majumder, Devamanyu Hazarika, Erik Cambria, Alexander Gelbukh, and Amir Hussain. Multimodal sentiment analysis: Addressing key issues and setting up the baselines. *IEEE Intelligent Systems*, 33(6):17–25, 2018.

Jacob Portes, Alexander Trott, Sam Havens, Daniel King, Abhinav Venigalla, Moin Nadeem, Nikhil Sardana, Daya Khudia, and Jonathan Frankle. Mosaicbert: a bidirectional encoder optimized for fast pretraining. *Advances in Neural Information Processing Systems*, 36, 2024.

Ofir Press, Noah A Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs. *arXiv preprint arXiv:2012.15832*, 2020.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

UEA Rik Henson. DecMeg2014 - Decoding the Human Brain. `https://www.kaggle.com/c/decoding-the-human-brain/data`, 2023. Accessed: 2024-04-15.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Hiroshi Shimodaira, Ken-ichi Noma, Mitsuru Nakai, and Shigeki Sagayama. Dynamic time-alignment kernel in support vector machine. *Advances in neural information processing systems*, 14, 2001.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. Generalizing dtw to the multi-dimensional case requires an adaptive approach. *Data mining and knowledge discovery*, 31:1–31, 2017.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Chenxi Sun, Yaliang Li, Hongyan Li, and Shenda Hong. Test: Text prototype aligned embedding to activate llm's ability for time series. *arXiv preprint arXiv:2308.08241*, 2023.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. *arXiv preprint arXiv:2106.00750*, 2021.

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

Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*, 2023.

Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26:275–309, 2013.

Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Etsformer: Exponential smoothing transformers for time-series forecasting. *arXiv preprint arXiv:2202.01381*, 2022.

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.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The eleventh international conference on learning representations*, 2022a.

Haixu Wu, Jialong Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Flowformer: Linearizing transformers with conservation flows. *arXiv preprint arXiv:2202.06258*, 2022b.

Hao Xue and Flora D. Salim. Promptcast: A new prompt-based learning paradigm for time series forecasting. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14, 2023. doi: 10.1109/TKDE.2023.3342137.

Chao-Han Huck Yang, Yun-Yun Tsai, and Pin-Yu Chen. Voice2series: Reprogramming acoustic models for time series classification. In *International conference on machine learning*, pp. 11808–11819. PMLR, 2021.

Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Ijcai*, volume 15, pp. 3995–4001. Buenos Aires, Argentina, 2015.

Xinli Yu, Zheng Chen, Yuan Ling, Shujing Dong, Zongyi Liu, and Yanbin Lu. Temporal data meets llm–explainable financial time series forecasting. *arXiv preprint arXiv:2306.11025*, 2023.

Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8980–8987, 2022.

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.

Tianping Zhang, Yizhuo Zhang, Wei Cao, Jiang Bian, Xiaohan Yi, Shun Zheng, and Jian Li. Less is more: Fast multivariate time series forecasting with light sampling-oriented mlp structures. *arXiv preprint arXiv:2207.01186*, 2022.

Xiyuan Zhang, Ranak Roy Chowdhury, Rajesh K. Gupta, and Jingbo Shang. Large language models for time series: A survey. *ArXiv*, abs/2402.01801, 2024. URL `https://api.semanticscholar.org/CorpusID:267411923`.

Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28 (1):162–169, 2017.

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.

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.

Tian Zhou, Peisong Niu, Liang Sun, Rong Jin, et al. One fits all: Power general time series analysis by pretrained lm. *Advances in neural information processing systems*, 36, 2024.

A Appendix

Table of Contents

A Related Works

In this Section, we review the related works in three key areas: time series classification, the application of language models to time series data, and text embeddings.

A.1 Time Series Classification

Time series classification has been an active research area for decades. Early approaches investigated distance-based approaches Abanda et al. (2019) for time series classification. Some built nearest neighbor classifiers based on explicit time series distance measures such as Dynamic Time Warping (DTW) Wang et al. (2013); Jeong et al. (2011); Berndt & Clifford (1994). Others have used distance kernels instead and learned Support Vector Machines (SVMs) Kampouraki et al. (2008); Bahlmann et al. (2002); Shimodaira et al. (2001), or extracted features and learned linear Dempster et al. (2020) or tree-based classifiers such as eXtreme Gradient Boosting (XGBoost) Chen & Guestrin (2016).

Later, deep learning-based approaches are widely adopted because of their ability to learn complex patterns. Convolutional Neural Networks (CNNs) have proven to be successful in learning local patterns in time series data Wu et al. (2022a); Franceschi et al. (2019); Zhao et al. (2017). Similarly, Multilayer Perceptron (MLP) can provide simple but effective time series classifiers Zhang et al. (2022). Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) effectively handle long sequence modeling Gu et al. (2021); Lai et al. (2018); Hochreiter & Schmidhuber (1997).

More recently, transformer-based models Vaswani et al. (2017) have revolutionized the NLP domain, and these models have been adapted to the time series domain Zhou et al. (2022); Wu et al. (2021); Zhou et al. (2021). The self-attention mechanism in transformers is known for modeling long-range dependencies in sequence data. However, the increasing complexity of these models often comes with larger model sizes and higher computational costs, especially for training.

A.2 Language Models for Time Series

The success of language modeling in NLP and LLMs has inspired researchers to harness LLMs in the time series domain. Comprehensive surveys Zhang et al. (2024); Jiang et al. (2024) have offered valuable insights into the integration of LLMs in time series analysis, highlighting key methodologies, challenges, and future directions. Gruver et al. (2023); Xue & Salim (2023); Cao et al. (2023) enabled pre-trained LLMs to generate time series forecasts through prompting. Further, Yu et al. (2023) delved into the potential of LLMs for generating explainable forecasts of financial time series. Jin et al. (2023) introduced Time-LLM, which focuses on learning to project time series into the language embedding space and directly using pre-trained LLM for time series forecasting tasks Yang et al. (2021). More importantly, recent work, OneFitsAll by Zhou et al. (2024), has shown promising results by fine-tuning models like GPT Radford et al. (2019) on time series tasks. OneFitsAll achieved SOTA performance across various time series tasks, including classification. In contrast, we propose using text embeddings instead of LLMs for time series analysis. We found that this approach leads to new SOTA performance on time series classification tasks, with the advantage of using far fewer trainable parameters than OneFitsAll.

A.3 Text Embeddings

Text embeddings have played a crucial role in NLP. These embeddings map words or sentences into a dense vector space, capturing semantic and syntactic information. Various text embedding techniques have been proposed, ranging from word-level embeddings like Word2Vec Mikolov et al. (2013) and GloVe Pennington et al. (2014) to contextualized embeddings obtained from pre-trained language models such as BERT Devlin et al. (2018) and RoBERTa Liu et al. (2019b). In the time series domain, some works have proposed unsupervised methods for learning time series embeddings Sun et al. (2023); Yue et al. (2022); Franceschi et al. (2019); Tonekaboni et al. (2021). However, the availability of large-scale datasets in the time series domain is generally more limited compared to those in the NLP domain, making learning time series embedding from scratch more challenging compared to text embeddings. To our best knowledge, we are the **first** to leverage the well-trained text embeddings from the NLP domain for time series classification.

B Datasets

We benchmark our model using the following 10 multivariate datasets from the UEA Time Series Classification Archive Bagnall et al. (2018). See Table 3 for their data characteristics.

Table 3: Dataset Characteristics. **Abbreviations:** EC: EthanolConcentration, FD: FaceDetection, HW: Handwriting, HB: Heartbeat, JV: JapaneseVowels, PEMS: PEMS-SF, SCP1: SelfRegulationSCP1, SCP2: SelfRegulationSCP2, SAD: SpokenArabicDigits, UW: UWaveGestureLibrary

Characteristic	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW
Train Size	261	5890	150	204	270	267	268	200	6599	120
Test Size	263	3524	850	205	370	173	293	180	2199	320
Number of Dimensions	3	144	3	61	12	963	6	7	13	3
Series Length	1751	62	152	405	29	144	896	1152	93	315
Number of Classes	4	2	26	2	9	7	2	2	10	8
Type	Spectro	EEG	Motion/Human Activity Recognition	Audio	Audio	Occupancy rate	EEG	EEG	Speech	EEG

B.1 EthanolConcentration

EthanolConcentration (Large et al., 2018) comprises raw spectra from water-and-ethanol solutions contained within 44 unique, real whisky bottles, featuring ethanol concentrations of 35%, 38%, 40%, and 45%. Scotch Whisky regulations require a minimum alcohol content of 40%, a standard that producers adhere to in order to comply with labeling specifications. The dataset presents a classification task to identify the ethanol concentration from spectral readings of any given bottle. Each record includes three spectral readings from the same bottle and batch, obtained by positioning the bottle between a light source and a spectroscope. These spectral readings, which cover wavelengths from 226nm to 1101.5nm at a 0.5nm resolution, were recorded over a one-second integration time using a StellarNet BLACKComet-SR spectrometer. The methodology deliberately avoids optimizing for clarity or consistency in the spectral path, aiming to simulate the varied conditions typical of rapid screening tests that may be performed on batches of spirits for quality assurance.

B.2 FaceDetection

The FaceDetection dataset originates from a 2014 Kaggle competition (Rik Henson, 2023). The challenge involves identifying whether a subject is viewing a picture of a face or a scrambled image using magnetoencephalography (MEG) data, independent of the individual subject. This dataset specifically includes only the training portion from the competition, organized by patient. It comprises data from 10 training subjects (subject01 to subject10) and 6 testing subjects (subject11 to subject16). Each subject has approximately 580 to 590 trials, resulting in a total of 5,890 training trials and 3,524 test trials. Each trial features 1.5 seconds of MEG data, initiated 0.5 seconds before the stimulus is presented, and is associated with a class label—Face (class 1) or Scramble (class 0). The data were down-sampled to 250Hz and subjected to a high-pass filter at 1Hz, producing 62 observations per channel.

B.3 Handwriting

The Handwriting dataset (Shokoohi-Yekta et al., 2017) consists of motion data captured from a smartwatch while subjects wrote the 26 letters of the alphabet. Developed at the University of California, Riverside (UCR), this dataset includes 150 training cases and 850 test cases. It features six dimensions, comprising three accelerometer readings and three gyroscope readings.

B.4 Heartbeat

The Heartbeat dataset originates from the PhysioNet/CinC Challenge 2016 Liu et al. (2016) and consists of cardiac sound recordings from a diverse pool of participants, both healthy individuals and patients with cardiac conditions. Recordings were made in various settings, clinical and non-clinical, and captured from multiple body locations including the aortic, pulmonic, tricuspid, and mitral areas, among up to nine potential sites. The dataset categorizes these sounds into two primary classes: normal and abnormal. Normal heart sounds were obtained from healthy subjects, while abnormal sounds were recorded from patients diagnosed with cardiac ailments, predominantly heart valve defects such as mitral valve prolapse, mitral regurgitation, aortic stenosis, and post-valvular surgery conditions, as well as coronary artery disease.

The audio recordings, inclusive of contributions from both children and adults, were uniformly truncated to five seconds. Spectrograms of each truncated audio were generated using a window size of 0.061 seconds with a 70% overlap. This multivariate dataset is structured with each dimension representing a frequency band derived from the spectrogram. There are 113 instances in the normal class and 296 in the abnormal class.

B.5 JapaneseVowels

The Japanese Vowels dataset Kudo et al. (1999), sourced from the UCI Machine Learning Repository, comprises recordings from nine male speakers who pronounced the Japanese vowels 'a' and 'e'. Each utterance was analyzed using a 12-degree linear prediction to extract a 12-dimensional time-series representation, with lengths varying originally from 7 to 29. For consistency, all instances in the dataset have been padded to the maximum length of 29. The objective of the classification task is to identify the speaker; hence each 12-by-29 instance matrix is associated with a single class label, ranging from 1 to 9. This dataset serves as a benchmark for assessing the efficacy of time-series classification models in distinguishing speakers based on LPC cepstrum coefficients obtained from their speech patterns.

The dataset includes a total of 640 time-series instances. A training set consists of 30 utterances per speaker, totaling 270 instances. The test set, however, comprises 370 instances and varies in distribution—ranging from 24 to 88 instances per speaker—owing to external factors such as timing and availability during the experimental setup.

B.6 PEMS-SF

The PEMS-SF dataset Cuturi (2011) contains 15 months of daily data sourced from the California Department of Transportation. This dataset details the occupancy rates, ranging from 0 to 1, across various car lanes on the freeways of the San Francisco Bay Area. The data spans from January 1, 2008, to March 30, 2009, with measurements taken every 10 minutes. Each day is treated as an individual time series with a dimension of 963, corresponding to the number of sensors that consistently functioned throughout the observation period. The length of each time series is 144 data points (6 per hour x 24 hours). The dataset excludes public holidays and two anomalous days (March 8, 2009, and March 9, 2008) when sensors recorded no data between 2:00 and 3:00 AM, resulting in a total of 440 valid time series. The classification task involves identifying the day of the week for each series, labeling them with integers from 1 (Monday) to 7 (Sunday). Each attribute within a record reflects the occupancy rate recorded by a sensor at a specific timestamp throughout the day.

B.7 SelfRegulationSCP1

The SelfRegulationSCP1 dataset, sourced from Birbaumer et al. (1999), involves recordings from a healthy subject who was instructed to control a cursor on a screen using cortical potentials. This process was facilitated by tracking the subject's slow cortical potentials (Cz-Mastoids), where cortical positivity resulted in downward cursor movements and cortical negativity caused it to move upward. Each trial, lasting six seconds, was designed to capture these dynamics, with visual feedback provided between the second 2 and 5.5 of the trial. During each trial, a goal was visually indicated at either the top or bottom of the screen starting from 0.5 seconds to the end of the trial, guiding the subject to generate negative or positive potentials correspondingly. The usable data for each trial, however, spans only 3.5 seconds—from the second 2 to 5.5—corresponding to 896 samples per channel given the sampling rate of 256 Hz.

Data capture involved a PsyLab EEG8 amplifier and a PCIM-DAS1602/16 A/D converter, recording over channels positioned according to the 10/20 system. The dataset includes a training set of 268 trials—168 from the first day and 100 from the second, mixed randomly—and 293 test instances, with class labels indicating positivity or negativity.

B.8 SelfRegulationSCP2

The SelfRegulationSCP2 dataset Birbaumer et al. (1999) includes data from an artificially respirated ALS patient who was tasked with controlling a cursor on a computer screen using cortical potentials. Auditory and visual cues were used to guide the patient, with slow cortical potentials measured at the Cz-Mastoids. A positive potential moved the cursor downward, whereas a negative potential moved it upward. Each trial lasted 8 seconds, with the cursor movement direction (up for negativity, down for positivity) indicated both visually and auditorily from the 0.5 to 7.5 second marks. Auditory instructions were given precisely at the 0.5-second mark, and visual feedback was available from seconds 2 to 6.5. Only the data from this 4.5-second feedback period, translating to 1152 samples per channel at a 256 Hz sampling rate, are used for training and testing.

EEG data were collected from several sites according to the 10/20 system and included channels for detecting vertical eye movements (vEOG). The EEG signals were not corrected for EOG artifacts, providing a raw view of the cortical activity. The dataset comprises 200 trials for training, evenly split between two classes, and an additional 180 trials for testing, recorded on the same day but after the training session data. Each trial spans 7 dimensions and a series length of 1152.

B.9 Spoken Arabic Digits

The Spoken Arabic Digits dataset Bedda & Hammami (2010) consists of 8,800 time series data entries derived from the vocal utterances of 88 native Arabic speakers (44 males and 44 females, aged between 18 and 40). Each dataset entry represents one of ten Arabic digits, spoken ten times by each speaker. The dataset captures 13 Mel Frequency Cepstral Coefficients (MFCCs) for each sound snippet, which are extracted under the following audio processing conditions:

- **Sampling rate:** 11025 Hz
- **Bit depth:** 16 bits
- **Window function:** Hamming
- **Pre-emphasis filter:** $1 - 0.97Z^{-1}$

Each line in the database corresponds to one frame of analysis, listing the 13 MFCCs separated by spaces. These coefficients effectively capture the spectral properties essential for recognizing spoken digits. This structured approach facilitates robust time-series analysis for speech recognition tasks involving Arabic numerals.

B.10 UWaveGestureLibrary

The UWaveGestureLibrary Liu et al. (2009) comprises a set of eight simple gestures, each generated from accelerometer data. The dataset records the X, Y, and Z coordinates corresponding to each gesture's motion. Every time series within this dataset consists of 315 data points.

C Comparison Baselines

To provide a thorough comparison, we evaluate our approach against 20 baseline models for time series classification. These baselines can be categorized into Classical methods, models based on MLPs, RNNs, CNNs, transformers, and LLMs. The details of which are provided below.

C.1 Classical methods

1) **Dynamic Time Warping (DTW)** Berndt & Clifford (1994) is a method for measuring similarity between two time series, $X = (x_1, x_2, \ldots, x_M)$ and $Y = (y_1, y_2, \ldots, y_N)$, which may differ in length and are sampled at equidistant points in time. DTW identifies the best alignment between these series by minimizing the effects of distortion and shifting in time, allowing for the comparison of similar shapes across different phases.

   The core of DTW is the construction of a local cost matrix, $C \in \mathbb{R}^{M \times N}$, with entries $c_{i,j} = \|x_i - y_j\|$ for $i \in [1, M]$ and $j \in [1, N]$, which represents the pairwise distances between points in the two series. The objective is to find a warping path $p = (p_1, p_2, \ldots, p_L)$ where each $p_l = (p_i, p_j)$ lies within $[1, M] \times [1, N]$. This path aligns the series by following the route that minimizes cumulative distance, adhering to several constraints: it must start and end at $p_1 = (1, 1)$ and $p_L = (M, N)$ (boundary condition), maintain the temporal ordering of points $m_1 \leq m_2 \leq \ldots \leq m_L$ and $n_1 \leq n_2 \leq \ldots \leq n_L$ (monotonicity condition), and prevent large temporal jumps (step size condition). The optimal warping path is identified through a recursive process aimed at minimizing the total cost associated with p, calculated as $c_p(X, Y) = \sum_{l=1}^{L} c(x_{m_l}, y_{n_l})$. This path, P^*, where $c_{P^*} = \min_{p \in P} c_p(X, Y)$, defines the DTW distance, quantifying the similarity between the series. However, the computational cost of this process is $O(MN)$, where M and N are the lengths of the two series, rendering it computationally demanding for large datasets.

   For classifying time series, the DTW distance is integrated with the k-nearest neighbors (k-NN) algorithm. This approach computes the DTW distance of a target series to all other series in a training dataset and assigns a classification based on the most common class among the k-nearest neighbors. Thus, DTW effectively accommodates series with time shifts, providing a robust, distance-based method for classifying time series.

2) **eXtreme Gradient Boosting (XGBoost)** (Chen & Guestrin, 2016) is a state-of-the-art machine learning algorithm that primarily uses decision trees as base learners to construct a robust ensemble model. XGBoost sequentially builds a series of weak learners—typically decision trees—and enhances each successive tree by correcting errors made by its predecessors, a technique known as boosting. This process involves an additive model where each new decision tree is improved by leveraging the cumulative knowledge of the trees that came before it, optimizing for maximum information gain at each split using a greedy

algorithm. To prevent overfitting, XGBoost incorporates regularization directly into its loss function and employs shrinkage to moderate the learning rate. Additionally, the optimization method, which is gradient-based, minimizes a cost function by iteratively adjusting the model's parameters in response to the gradients of the errors. XGBoost also refines the decision tree construction process by using a Similarity Score and Gain to determine the most effective node splits, further improving the model's accuracy and efficiency.

3) **RandOm Convolutional KErnel Transform (ROCKET)** (Dempster et al., 2020) is a method for time series classification that employs random convolutional kernels to transform series data, which is then used to train a linear classifier. Unlike traditional convolutional neural networks (CNNs) that rely on learned kernels, ROCKET utilizes a broad array of random kernels. Each kernel is uniquely characterized by random properties such as length, weights, biases, dilation, and padding. This configuration forms a single-layer convolutional neural network, where the randomized kernel weights contribute to generating input for a softmax layer, thus optimizing the feature extraction process. ROCKET also efficiently scales for large datasets due to its linear complexity relative to the length of the time series and the number of training samples. The key advantages of ROCKET include:
   - Number of Kernels: ROCKET utilizes a substantial number of kernels in a single layer. The non-learned nature of these kernels reduces computational costs significantly, enabling the use of numerous kernels without substantial overhead.
   - Variety of Kernels: Unlike typical CNNs, where kernels might share characteristics, each ROCKET kernel is distinct in its attributes, enhancing the diversity and the ability to detect various patterns.
   - Kernel Dilation: Dilation in ROCKET is randomly assigned to each kernel, differing from the exponential increase with depth seen in traditional CNNs. This randomness is vital for identifying patterns across different scales and frequencies.
   - Feature Extraction: Beyond employing global max pooling techniques through the maximum value of feature maps, ROCKET utilizes a novel metric—the proportion of positive values. This metric allows classifiers to assess the prevalence of patterns more accurately within the time series.

C.2 MLP-based methods

4) **LightTS** (Zhang et al., 2022) is an MLP-based time series forecasting model that employs simple MLP structures to manage both short-term and long-term temporal dependencies. This model includes two downsampling strategies: interval sampling, which targets long-term dependencies, and continuous sampling, which focuses on short-term local patterns. These strategies are based on the principle that downsampling typically preserves the majority of a time series' crucial information, thus maintaining model efficiency. LightTS utilizes an MLP-based framework on top of these downsampling techniques, enabling effective information exchange among different down-sampled subsequences and time steps. This configuration allows LightTS to adaptively select relevant information for forecasting and to efficiently handle very long input sequences by processing only a fraction of the data after downsampling.

5) **DLinear** (Zeng et al., 2023) is a recent non-transformer model developed in response to the difficulty transformer-based models face in capturing ordering information within time series. DLinear integrates a decomposition scheme similar to those used in Autoformer and FEDformer but relies on linear layers for processing. Initially, it decomposes a raw data input into a trend component using a moving average kernel and a remainder (seasonal) component. Subsequently, two single-layer linear layers are applied independently to each component. The outputs from these layers are then summed to produce the final prediction. This approach allows DLinear to enhance performance over a standard linear model by explicitly handling trends within the data.

C.3 RNN-based models

6) **Long Short-Term Memory (LSTM)** Hochreiter & Schmidhuber (1997) addresses the vanishing gradient problem that plagues vanilla RNNs in processing longer sequences. This issue arises as the network propagates forward, and the small weight values in the

hidden layers are multiplied repeatedly, causing the gradients to diminish rapidly. As a result, the weights in the initial layers become increasingly difficult to train, which impacts the training of subsequent weights, making RNNs challenging to train overall. LSTM mitigates this problem by incorporating a memory cell equipped with various gates that regulate the flow of information into and out of the cell, enabling it to handle long-short term dependencies effectively. An LSTM unit utilizes a cell state and three gates—input, forget, and output—to manage information. Each gate includes a sigmoid layer σ that outputs values between 0 and 1, representing the proportion of information allowed through the gate, and a point-wise multiplication operation. Specifically, the forget gate $f_t = \sigma(W_f \cdot [h_{t-1},\, x_t] + b_f)$ determines which information to discard from the previous cell state c_{t-1} by analyzing the current input x_t and the previous hidden state h_{t-1}. The input gate $i_t = \sigma(W_i \cdot [h_{t-1},\, x_t] + b_i)$ decides which new information to update, and the update to the cell state $\tilde{c}_t = \tanh(W_c \cdot [h_{t-1},\, x_t] + b_c)$ is computed. The cell state is then updated to $c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$. Finally, the output gate $o_t = \sigma(W_o \cdot [h_{t-1},\, x_t] + b_o)$ determines what portion of the cell state to output, with the output hidden state given by $h_t = o_t \cdot \tanh(c_t)$.

7) **Long- and Short-term Time-series Network (LSTNet)** (Lai et al., 2018) incorporates both CNN and RNN components to perform comprehensive time series analysis. The CNN extracts short-term local dependency patterns from multi-dimensional input variables, while the RNN is tasked with capturing complex long-term dependencies in time series trends. To address the issue of scale insensitivity commonly found in neural network models, LSTNet integrates a traditional autoregressive model. Additionally, LSTNet features a Recurrent-skip structure designed to effectively capture very long-term dependence patterns and to facilitate easier optimization, leveraging the periodic properties of the input time series signals. Further enhancing its robustness, LSTNet also employs a traditional autoregressive linear model in parallel with its nonlinear neural network components. This dual approach makes the network particularly adept at managing time series that exhibit significant scale variations.

8) **Linear State Space Layer (LSSL)** (Gu et al., 2021), which is part of the structured state space sequence model, introduces a parameterization for state space models (SSM) designed to enhance computational efficiency. LSSL modifies the structured state matrices by decomposing them into a low-rank and a normal term Gu et al. (2020); Voelker et al. (2019). This modification facilitates the computation of the truncated generating function in frequency space, rather than expanding the standard SSM in coefficient space. Furthermore, LSSL employs the Woodbury identity to adjust the low-rank term, and the normal term is stably diagonalized. This approach simplifies the computations by involving processes associated with a Cauchy kernel Pan (2012, 2017), known for its stability in theoretical contexts. These modifications allow LSSL to efficiently manage both computational and memory resources.

C.4 CNN-based models

9) **Temporal Convolutional Network (TCN)** (Bai et al., 2018; Franceschi et al., 2019) is a CNN-based architecture designed to capture extended historical data with long memory capabilities and requires minimal tuning in practice. TCNs utilize dilated causal convolutions to ensure that predictions do not prematurely incorporate future data. The dilations significantly expand the network's receptive field, enabling it to cover a broader range of historical context. Furthermore, TCNs integrate residual connections to facilitate the effective training of deeper models.

   A TCN model comprises a series of n TCN residual blocks, where n is a hyperparameter. Each block contains two dilated causal convolutional layers, which are fully convolutional to ensure that the output size is consistent with the input size. These causal convolutions guarantee that the output at any given time t depends only on inputs from time t and earlier. The convolutional layers are applied with a stride of 1, and padding adjustments maintain the convolutional nature of the network. Each convolutional layer applies a dilation factor d, typically set as $d = 2^i$ for the i-th block, to exponentially increase the receptive field as the network deepens. Mathematically, a convolution with dilation factor d on an element x of a 1D input g with a filter f of length k is computed as $(g *_d f)(x) = \sum_{j=0}^{k-1} f(j) \cdot g(x - d \cdot j)$.

Following the convolutional layers, the sequence of operations includes weight normalization Salimans & Kingma (2016), ReLU activation, and a dropout layer. Weight normalization improves gradient conditioning and accelerates convergence by reparameterizing each weight vector $\mathbf{w}$ as $\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$, where $\mathbf{v}$ has a fixed norm and g is a scalar. This process decouples the magnitude of the weight vector from its direction, enhancing network optimizability. Each block concludes with an element-wise addition of the block's input and the estimated residual mapping, followed by a ReLU activation. Dimension alignment for this addition is achieved with a 1×1 convolution.

10) **TimesNet** (Wu et al., 2022a) is a method that overcomes the limitations of traditional 1D time series representation by transforming these series into 2D tensors organized across multiple periods. It captures short-term intraperiod variations and long-term interperiod trends by embedding them into the columns and rows of the 2D tensors, respectively. This transformation allows for more efficient modeling of temporal variations using 2D convolutional kernels, thereby extending the analysis into a more comprehensive 2D space. TimesNet ensures simultaneous representation of both intraperiod and interperiod variations, with modules specifically tailored to emphasize the unique temporal patterns of each period.

    The central component of TimesNet, the TimesBlock, is a versatile and adaptive structure designed to detect multiperiodicity and extract complex temporal patterns from these 2D tensors. Its parameter-efficient Inception-block (Szegedy et al., 2015) based architecture boosts the model's analytical capabilities, facilitating a detailed examination of distinct temporal variations associated with different periods. This approach allows TimesNet to transcend the constraints of 1D representations, enabling a unified and thorough analysis of temporal variations.

C.5 Transformer-based models

11) **Transformer** (Vaswani et al., 2017) comprises a dual-component architecture with both an encoder and a decoder, each containing stacked self-attention and point-wise, fully connected layers. Each component consists of six identical layers. In the encoder, each layer includes a multi-head self-attention mechanism and a position-wise fully connected feed-forward network, complemented by a residual connection and layer normalization. The decoder replicates this configuration but adds a third sub-layer for multi-head attention on the encoder's output. It also modifies its self-attention mechanism to prevent forward-looking attention, thus preserving the model's auto-regressive properties for sequential generation.

    The Transformer utilizes multi-head attention to enable nuanced interactions between its encoder and decoder and to facilitate detailed processing across different positions in the sequence. This attention mechanism projects queries, keys, and values through multiple linear transformations, enabling diverse representation and integration of information across subspaces. It is applied in three distinct forms: encoder-decoder attention allows decoder queries to attend to all positions in the encoder output; self-attention within the encoder lets each position process information from all preceding positions; and self-attention within the decoder restricts attention to prevent future positions from influencing the sequence, maintaining sequential integrity. This structured approach helps the Transformer effectively manage and process long sequence dependencies, making it adaptable for a broad range of sequence-based applications.

12) **Reformer** (Kitaev et al., 2020) enhances the efficiency of the transformer model with two significant modifications, making it more suitable for processing long sequences. Firstly, it replaces the traditional dot-product attention with a locality-sensitive hashing mechanism. This change reduces the computational complexity from $O(L^2)$ to $O(L \log L)$, where L is the sequence length. Secondly, Reformer employs reversible residual layers, which regenerate the activations of any layer from the subsequent layer's activations using only the model parameters. This approach eliminates the need for storing multiple copies of activations for each layer, substantially reducing memory usage. The reversible layers, introduced in Gomez et al. (2017), require only a single set of activations to be stored for the entire model, significantly reducing the memory cost usually multiplied by the number of layers (N). Moreover, the Reformer processes activations in chunks within feed-forward layers, further decreasing memory demands. These adjustments, along with the use of locality-sensitive hashing for attention computation, not only minimize memory

and computational overhead but also maintain the model's performance on par with the traditional transformer model for long sequences.

13) **Informer** (Zhou et al., 2021) is a transformer-based model designed specifically to tackle challenges in long sequence time-series forecasting, such as quadratic time complexity, high memory usage, and constraints of the traditional encoder-decoder architecture. The Informer introduces several modifications to enhance efficiency and effectiveness in processing long sequences:

    – ProbSparse Self-Attention Mechanism: This mechanism replaces the conventional self-attention in Transformers. It is engineered to achieve $O(L \log L)$ in both time complexity and memory usage, efficiently managing dependency alignments without compromising performance.
    – Self-Attention Distilling: This process improves attention management by concentrating on dominant attention scores and halving the input size for each cascading layer. This method effectively manages extremely long input sequences and significantly lowers the space complexity to $O((2 - \epsilon)L \log L)$.
    – Generative Style Decoder: Diverging from the typical step-by-step decoding process, the generative style decoder in Informer predicts long time-series sequences in a single forward operation. This approach accelerates inference for long-sequence predictions and reduces the propagation of cumulative errors during the inference phase.

    The Informer leverages these enhancements—ProbSparse self-attention for efficient processing, self-attention distilling to focus on important attention scores, and a generative style decoder for rapid sequence generation—to improve its performance in forecasting long sequences and capturing long-range dependencies between extensive time-series inputs and outputs.

14) **Pyraformer** (Liu et al., 2021) is a transformer-based model that employs a pyramidal attention module (PAM) for efficient management of time series data. This module uses inter-scale and intra-scale connections to summarize features at different resolutions and capture temporal dependencies across various ranges. The design integrates a tree structure for inter-scale connections and neighboring intra-scale connections to achieve a multi-resolution representation of time series. This architecture ensures that Pyraformer scales linearly with the input series length, optimizing computational efficiency while maintaining a constant signal path length relative to the sequence length L, and keeping both time and space complexity linear with L.

    Pyraformer operates by first embedding observed data, covariates, and positions in a manner similar to the Informer (Zhou et al., 2021). It then constructs a multi-resolution C-ary tree through a coarser-scale construction module (CSCM), where each coarser scale node aggregates information from C finer scale nodes. This structure allows Pyraformer to model temporal dependencies efficiently across different scales through sparse intra-scale connections, thus reducing computational overhead. Depending on the specific needs of different downstream tasks, Pyraformer adapts its output structure to effectively meet the requirements of diverse time series analyses.

15) **Autoformer** (Wu et al., 2021) is a transformer-based model that incorporates time series decomposition, drawing inspiration from classical time series analysis methods. Unlike traditional transformers that rely on self-attention mechanisms to capture long-range dependencies, Autoformer introduces an auto-correlation mechanism as an alternative. This change addresses the issues that traditional models face with intricate temporal patterns and long-term forecasting, where identifying reliable dependencies can be challenging. To enhance efficiency in handling long series, traditional transformers sometimes use sparse versions of self-attention, which can limit the effective use of information.

    Autoformer integrates decomposition blocks directly into its structure, moving away from the typical preprocessing approach of series decomposition. These blocks are specifically designed to progressively isolate long-term trends from the data during the forecasting process, allowing the model to refine and decompose the data iteratively. This setup improves the handling of complex time series. The auto-correlation mechanism, inspired by stochastic process theory and based on the periodicity of the series, focuses on identifying dependencies and aggregating representation at the sub-series level, effectively capturing and utilizing similarities derived from underlying periodic patterns.

The architecture of Autoformer adheres to a residual and encoder-decoder framework. The encoder eliminates long-term trend-cyclical components through the series decomposition blocks and focuses on modeling seasonal patterns. In contrast, the decoder accumulates the trend component extracted from the hidden variables, using past seasonal information to enhance forecasting accuracy. Additionally, Autoformer incorporates a moving average within its decomposition blocks to smooth out periodic fluctuations and highlight long-term trends, thereby facilitating a more targeted analysis of stable trend components within the time series.

16) **Stationformer** (Liu et al., 2022) addresses the challenges posed by non-stationary real-world data, where the joint distribution changes over time, often leading to the degradation of transformer performance. Stationformer consists of two interdependent modules: series stationarization and de-stationary attention. Series stationarization normalizes the input data to unify its statistical properties, enhancing predictability, and adjusts the output to restore the original statistics. This module utilizes a straightforward normalization approach without additional parameters. De-stationary attention, on the other hand, aims to counteract the potential over-normalization by reintroducing the intrinsic non-stationary characteristics of the data into the model's temporal dependencies. The de-stationary attention module approximates how attention mechanisms would function on unnormalized data and integrates these insights back into the model to maintain crucial temporal dynamics. This setup allows stationformer to balance the predictability benefits gained from normalized data with the rich, detailed patterns inherent in the raw non-stationary series.

    Structurally, Stationformer adapts the traditional encoder-decoder setup. The encoder extracts information from past observations, while the decoder aggregates this information to refine predictions. The framework modifies the standard transformer by applying series stationarization to both the input and output of the model, and replaces the conventional self-attention mechanism with de-stationary attention. This adaptation aims to enhance the model's ability to predict non-stationary series by effectively managing the challenges associated with data variability over time.

17) **Frequency Enhanced Decomposed Transformer (FEDformer)** (Zhou et al., 2022) is a transformer-based method that incorporates a time series decomposition scheme to address the limitations of traditional transformers, particularly their high computational demands and challenges in capturing global time series trends. By combining transformers with the seasonal-trend decomposition method, FEDformer aims to separate the broad trends from more detailed fluctuations in time series data. This allows the transformer component to focus on more granular details while the decomposition handles the overall profile of the series.

    The architecture of FEDformer includes specialized blocks such as Fourier-enhanced and Wavelet-enhanced blocks within the transformer framework. These blocks serve as substitutes for the conventional self-attention and cross-attention mechanisms, and they enable the model to analyze important structures through frequency domain mapping. FEDformer employs a selective approach to incorporating Fourier components, which helps keep the computational complexity and memory usage linear in relation to the length of the time series. Specifically, FEDformer is structured as a deep decomposition architecture that integrates Frequency Enhanced Block (FEB), Frequency Enhanced Attention (FEA) connecting the encoder and decoder, and the Mixture Of Experts Decomposition block (MOEDecomp). This setup leverages both seasonal-trend decomposition and distribution analysis to facilitate the processing of time series data.

18) **ETSformer** (Woo et al., 2022) is a transformer-based architecture tailored for time series forecasting, integrating exponential smoothing techniques. ETSformer uses two novel mechanisms, Exponential Smoothing Attention (ESA) and Frequency Attention (FA), which are designed to replace the traditional self-attention mechanism in standard transformers. ESA utilizes attention scores based on relative time lags, enabling efficient handling of growth components with a computational complexity of $O(L \log L)$ for a length-L lookback window. Similarly, FA employs Fourier transformations to identify dominant seasonal patterns, selecting bases with the highest amplitudes in the frequency domain to achieve the same level of complexity.

    ETSformer is structured with modular decomposition blocks that allow it to dissect time-series data into distinct, interpretable components such as level, growth, and seasonality.

This design facilitates layer-wise decomposition of the time series into these components, enhancing the model's ability to capture and represent complex temporal dynamics. The architecture systematically extracts latent growth and seasonal patterns through a deep, multi-layered approach, where each layer progressively refines the extraction of these temporal features. The final forecast generated by ETSformer integrates these decomposed elements—level, trend, and seasonality—into a cohesive output that is both practical and interpretable for human analysts. By emphasizing recent observations, the model aligns with the principles of exponential smoothing, ensuring that more recent trends carry greater weight in the forecast.

19) **Flowformer** (Wu et al., 2022b) modifies the traditional transformer architecture by integrating flow network theory to tackle the scalability issues typical of standard transformers. The conventional attention mechanism in transformers is known for its quadratic complexity, which limits their ability to process a large number of tokens and scale to larger models effectively. To address this, Flowformer introduces the Flow-Attention mechanism, grounded in the principles of flow conservation. This mechanism reimagines attention as information flowing from sources (values) to sinks (results) via learned flow capacities (attentions), aiming to achieve linear complexity.

    The Flow-Attention mechanism manages the flow of information by regulating incoming flow at sinks to initiate source competition and outgoing flow at sources for sink allocation. This management of flow helps in aggregating relevant information without relying on specific inductive biases and aims to prevent the common issue of degenerated attentions found in typical attention mechanisms. Flowformer embeds flow conservation within its attention mechanism to streamline the process of information aggregation and refinement. By integrating these elements, Flowformer seeks to provide an alternative approach that could potentially handle large datasets and complex time series data more efficiently than traditional transformers, without the computational complexity typically associated with these models.

C.6 LLM-based models

20) **OneFitsAll** (Zhou et al., 2024) employs pre-trained language and computer vision models, developed from billions of tokens, for time series analysis. This approach, termed the Frozen Pretrained Transformer (FPT), retains the original self-attention and feedforward (FFN) layers of the pre-trained models, which hold the majority of the learned knowledge. The model is fine-tuned for various time series classification tasks, with adjustments made only to the positional embeddings and layer normalization layers to adapt to specific downstream tasks. Additionally, to more effectively manage local semantic information, OneFitsAll incorporates a patching technique as described by Nie et al. (2022). This method aggregates adjacent time steps into a single patch-based token, thereby increasing the historical time horizon that can be processed by the model without increasing the token length, reducing information redundancy within transformer models.

D Hyperparameter Settings

D.1 Hyperparameter Search Space

Our hyperparameter search for the lightweight classification head included determining the optimal number of convolutional layers, whether to use dropout or pooling within the convolutional blocks, the number of dense layers, the type of activation function for each layer, and whether to employ batch normalization. We explored 1 to 5 convolutional layers and 1 to 4 linear layers. We also tested various learning rates, ranging from 0.001 to 0.05, different activation functions such as ReLU, GELU, and tanh, and truncation lengths for the text-embedding-large from 64 to 1024.

D.2 Optimal Hyperparameters

Table 4 presents the optimal configurations for our experiments across various datasets. All configurations employed tokenization with a precision of one decimal place and utilized the `text-embedding-3-large` embeddings. It is crucial to note that the count of linear layers includes the output layer; therefore, a configuration with one linear layer means that this single layer functions as the output layer within the model architecture. Furthermore, all convolutional layers utilized a kernel size of 3. Additionally, each configuration used the RAdam optimizer Liu et al. (2019a) with its default hyperparameter settings $(\beta_1, \beta_2) = (0.9, 0.999)$.

Table 4: Optimal hyperparameter configuration.

LETS-C						
Dataset/Configuration	Model Hyperparameter				Training Process	
	Embedding dimension	Conv layers	Linear layers	Activation	Learning rate	Batch size
EthanolConcentration	64	2	1	tanh	0.007	64
FaceDetection	64	1	1	tanh	0.007	128
Handwriting	16	1	2	tanh	0.007	64
Heartbeat	512	1	1	tanh	0.007	64
Japanese Vowels	512	1	2	GELU	0.007	64
PEMS-SF	1024	3	1	tanh	0.007	64
Self-Regulation SCP1	64	1	1	tanh	0.007	64
Self-Regulation SCP2	1024	2	1	GELU	0.007	64
Spoken Arabic Digits	512	1	1	tanh	0.001	64
UWave Gesture Library	1024	1	1	tanh	0.001	64

E Model Performance

Figure 3 displays a comparison of models based on the average classification accuracy across all datasets, as summarized in Table 3.

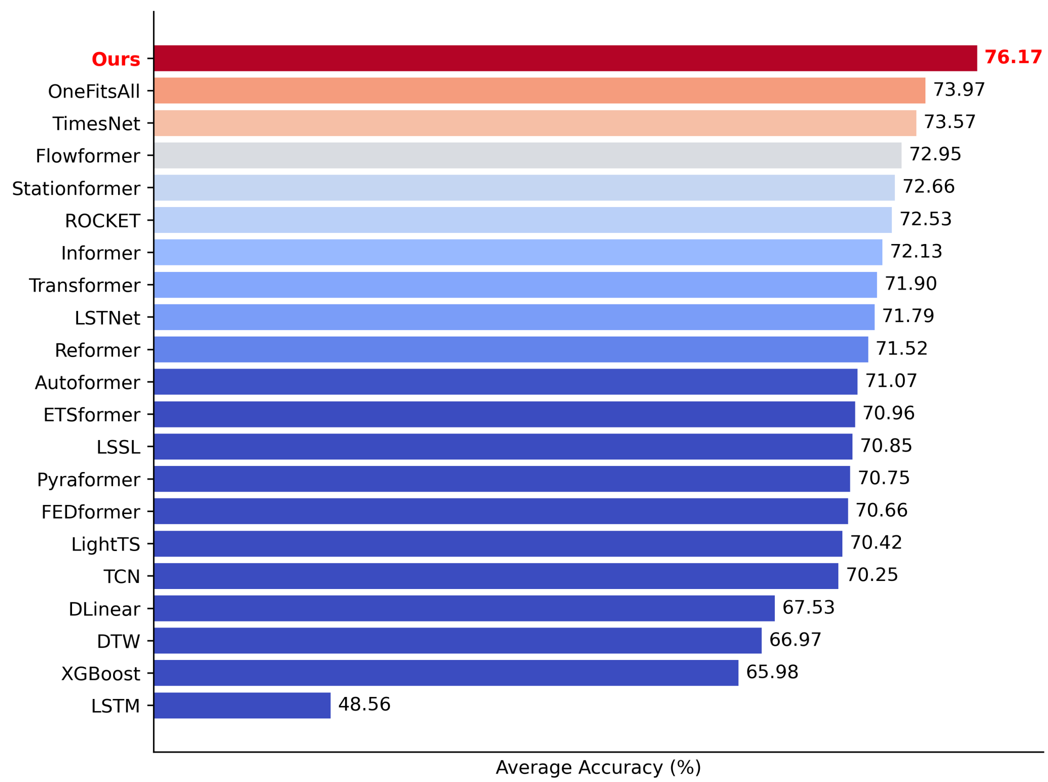

Figure 3: Model comparison based on classification accuracy, averaged across all datasets listed in Table 3. For detailed results, refer to Table 1 in the main paper.

F Computational Cost Analysis

Table 5 discusses the training and inference times for our model as compared to the SOTA OneFitsAll (Zhou et al., 2024) across the 10 benchmark datasets. We observe that our approach significantly

Table 5: Comparison of training and inference times (in seconds) per batch for OneFitsAll (Zhou et al., 2024) versus our model across the 10 benchmark datasets. The Ratio (%) is calculated using the formula $100 \times \frac{\text{time of LETS-C}}{\text{time of OneFitsAll}}$, quantifying the computational efficiency of our model relative to OneFitsAll.

	Model/Dataset	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↓**
Time												
Training time (s)	LETS-C (Ours)	0.01	0.01	0.009	0.01	0.01	0.11	0.004	0.01	0.01	0.01	0.02
	OneFitsAll	0.35	0.23	0.14	0.07	0.02	0.28	0.11	0.13	0.97	0.07	0.24
	Ratio (%)	3.17	8.02	6.71	16.18	42.85	40.49	4.30	8.40	1.21	15.30	**14.66**
Inference time (s)	LETS-C (Ours)	0.01	0.05	0.01	0.01	0.01	0.10	0.004	0.01	0.02	0.01	0.02
	OneFitsAll	0.16	0.10	0.06	0.04	0.02	0.15	0.06	0.08	0.15	0.04	0.09
	Ratio (%)	10.58	50.50	14.75	31.15	68.41	66.10	7.02	19.59	18.00	30.55	**31.67**

reduces the total training time to just 14.66% and the inference time to 31.67% of those required by OneFitsAll.

G Ablation Study

To empirically assess the advantages of fusing both text embeddings and time series data, as opposed to variants that only leverage either the text embedding or time series itself, we conducted an ablation study. Table 6 presents the results of this study, comparing the performance of our default approach, which adds embeddings to time series, to variants that exclude either embeddings or the time series.

Table 6: Comparison of classification accuracy (%) for different configurations: ours (both embeddings and time series), embeddings only, and time series only. **Red**: Best.

Method/Dataset	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↑**	**AvgWins % ↑**
LETS-C (Ours)	52.9	68.9	23.8	78	99.2	93.1	93.2	62.8	99.2	90.6	**76.17**	**60%**
embedding only	38	59.4	20.1	80	99.2	92.5	88.7	63.9	99.2	88.4	72.94	40%
time series only	42.6	69.6	25.1	76.1	98.1	89	93.2	58.3	98.9	83.8	73.47	30%

H Alternative Methods for Fusing Time Series with Embeddings

We explored two additional methods for fusing time series and embeddings beyond simple addition. The first method involves a *Fusion network* that first processes embeddings and time series data through convolutional and dense layers in two separate branches, then merges the features from both branches into a final dense network. The second method employs *Concatenation*, where the time series and embeddings are concatenated and processed through a lightweight classification head. Despite cross-attention being another alternative for fusing different modalities, we didn't include it in this study due to the computational complexity it adds to the model. Table 7 presents the classification accuracy and trainable model parameters for these variations.

I Assessing Text Embeddings with Cosine Similarity

Figure 4 visualizes the within-class and between-class cosine similarities of text embeddings derived from the testing time series. Each matrix entry is scaled using min-max normalization to range from 0 to 1, where warmer colors in the heatmap represent higher similarities and darker shades indicate lower similarities. Diagonal entries show within-class similarities, highlighting intra-class cohesion, while off-diagonal entries reveal between-class relationships

Table 7: Comparison of classification accuracy (%) and trainable model parameters (millions) for alternative methods for fusing time series with embeddings. Higher AvgWins and averages signify superior performance, while lower averages suggest greater computational efficiency. **Red:** Best performance.

Accuracy ↑												
Method/Dataset	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↑**	**AvgWins % ↑**
LETS-C (Addition)	52.9	68.9	23.8	78	99.2	93.1	93.2	62.8	99.2	90.6	**76.17**	**70%**
Fusion network	44.1	66.5	23.6	76.6	98.1	86.1	92.8	56.1	99.2	90.9	73.40	20%
Concatenation	43	65.1	22.5	79	98.9	93.1	93.9	58.9	99	88.8	74.22	30%

Trainable Parameters (M) ↓											
Method/Dataset	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↓**
LETS-C (Addition)	0.28	0.003	0.15	0.04	0.14	0.56	0.30	0.33	0.14	0.26	**0.22**
Fusion Network	0.19	0.35	0.33	0.28	0.16	5.54	0.37	0.27	0.23	0.04	0.78
Concatenation	0.42	0.009	0.26	0.17	0.11	0.28	0.23	0.39	0.09	0.46	0.24

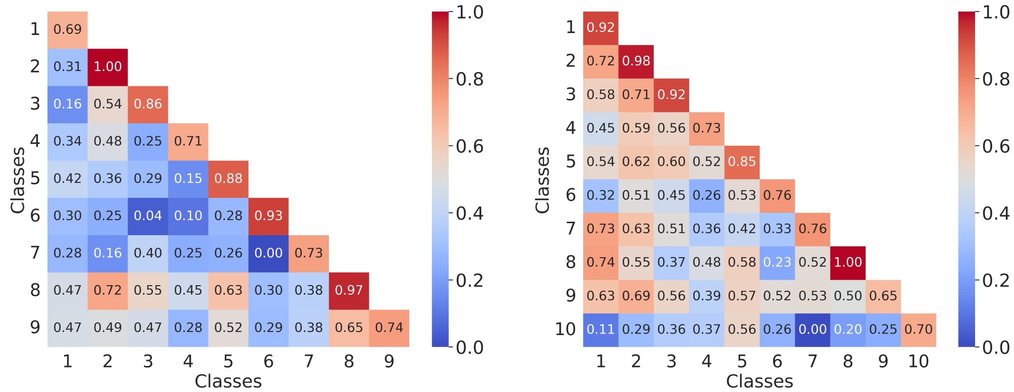

Figure 4: Heatmaps illustrating within-class and between-class cosine similarities of text embeddings derived from the testing time series data in Japanese Vowels **(left)** with 9 classes and Spoken Arabic Digits **(right)** with 10 classes. On both axes, x and y represent different classes. Diagonal entries indicate within-class similarities, and off-diagonal entries represent between-class similarities. Warmer colors signify higher cosine similarities, while cooler colors suggest lower similarities.

Similar to the training set, we observe that within-class similarity consistently exceeds across-class similarity, thereby validating the hypothesis that text embeddings effectively retain and convey significant information from the underlying time series data.

J Numerical Precision for Tokenization

To explore the impact of numerical precision on the computation of embeddings, we analyzed classification accuracy across precisions 1 to 6 using four datasets: Handwriting, Heartbeat, JapaneseVowels, and UWaveGestureLibrary. We selected these datasets because they were the smaller ones among the 10 available, making the computation of embeddings more computationally affordable. Figure 5 illustrates the average classification accuracy (%) across these numerical precisions. Detailed results can be found in Table 8.

Studies in the NLP domain have shown that longer inputs do not perform well with language models Press et al. (2020); Levy et al. (2024). Our empirical analysis supports this claim, revealing a decrease in classification accuracy with increased numerical precision when computing text embeddings. The average accuracy starts at 72.8% with precision 1 and declines to 69% at precision 6. Additionally, the percentage of AvgWins is highest at precision 1. As numerical precision increases, so does the length of the time series and the input to the text embedding model. Note that the maximum token length for `text-embedding-3-large` embeddings is 8191, and thus keeping precision of 1 ensures that context length doesn't exceed the maximum permissible token length. This issue is especially problematic for datasets with longer time series, such as the EthanolConcentration dataset, which includes 1751 time steps per sample. This finding led us to opt for precision 1 in our study.

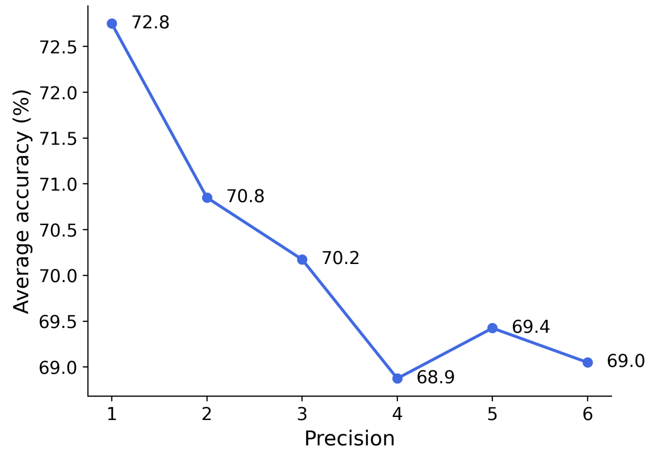

Figure 5: Average classification accuracy (%) across numerical precisions 1 to 6. Results are averaged from four datasets: Handwriting, Heartbeat, JapaneseVowels, and UWaveGestureLibrary. See Table 8 in the Appendix for detailed results.

Table 8: Numerical Precision for Tokenization: This table reports classification accuracy (%). **Red:** Best performance.

Dataset / Precision	1	2	3	4	5	6
Handwriting	23.2	15.9	11.6	11.1	12.0	11.2
Heartbeat	78.0	78.5	79.0	78.5	79.5	78.0
JapaneseVowels	99.2	98.4	97.6	96.8	95.9	95.1
UWaveGestureLibrary	90.6	90.6	92.5	89.1	90.3	91.9
Average ↑	**72.75**	70.85	70.175	68.875	69.425	69.05
AvgWins % ↑	**50%**	0%	25%	0%	25%	0%

Note that these precision results also depend on the type of tokenization selected, and defining an appropriate tokenization for time series is one of the potential future directions for this work.

K Embedding Type

To evaluate the generalization capabilities of our approach across different text embedding models beyond `text-embedding-3-large`, the performance of LETS-C was tested using three alternative embedding models: `e5-mistral-7b-instruct` Wang et al. (2023), `gte-large-en-v1.5` Li et al. (2023), and `nomic-embed-text-v1` Nussbaum et al. (2024). A summary of the embedding models utilized in this study is provided in Table 9.

Table 9: Summary of selected embedding models used in the study.

MTEB Rank	**Model**	**Embedding Dimensions**	**Max Token Length**
15	`text-embedding-3-large` OpenAI (2024)	3072	8191
6	`e5-mistral-7b-instruct` Wang et al. (2023)	4096	32768
9	`gte-large-en-v1.5` Li et al. (2023)	1024	8192
35	`nomic-embed-text-v1` Nussbaum et al. (2024)	768	8192

K.1 `e5-mistral-7b-instruct`

The `e5-mistral-7b-instruct` model Wang et al. (2023) is based on the pretrained Mistral-7b checkpoint Jiang et al. (2023) and was fine-tuned using the RankLLaMA training methodology Ma et al. (2024). It employed LoRA Hu et al. (2021) with a rank of 16 on a mixture of multilingual datasets, which included both synthetic data and a collection of 13 public datasets, yielding approximately 1.8 million examples after sampling. Proprietary LLMs, such as GPT-4, were prompted to generate diverse synthetic data with instructions in multiple languages. Leveraging the strong language understanding capabilities of the Mistral model, `e5-mistral-7b-instruct` achieved state-of-the-art results across nearly all task categories on the competitive MTEB benchmark. Techniques such as gradient checkpointing, mixed precision training, and DeepSpeed ZeRO-3 were applied to further reduce GPU memory requirements.

K.2 `gte-large-en-v1.5`

The `gte-large-en-v1.5` model Li et al. (2023) is a general text embedding (GTE) model that utilizes contrastive learning on an open-source large-scale dataset comprising unsupervised text pairs extracted from various sources. To enhance the quality of the learned text representations, high-quality text pairs with human labels from multiple sources were employed for contrastive fine-tuning. The model utilizes a multi-stage contrastive learning approach and benefits from a diverse training data mixture, enabling it to achieve strong generalization performance for single-vector embeddings.

K.3 `nomic-embed-text-v1`

The `nomic-embed-text-v1` model Nussbaum et al. (2024) is a fully reproducible, open-source English text embedding model with an 8192 context length that outperforms both OpenAI Ada-002 and OpenAI text-embedding-3-small on short and long-context tasks. To accommodate long sequence lengths, the model adapts the BERT architecture Devlin et al. (2018) with several optimizations: replacing absolute positional embeddings with rotary positional embeddings Su et al. (2024), using SwiGLU activation instead of GeLU Shazeer (2020), implementing Flash Attention Dao et al. (2022), setting Dropout to 0 Geiping & Goldstein (2023), and ensuring the vocabulary size is a multiple of 64 Portes et al. (2024); Shoeybi et al. (2019). These modifications result in a 137M parameter encoder.

Table 10 presents detailed accuracy metrics and trainable parameters for these various embedding models in the LETS-C framework. Note that among these, `e5-mistral-7b-instruct`

Table 10: Comparison of LETS-C with various embedding models against OneFitsAll. Performances surpassing OneFitsAll are in **bold**, with the best in **Red**. AvgWins scores above 50% indicate consistent superiority, calculated as 1 for outperforming OneFitsAll and 0 otherwise. Ratio (%) $= 100 \times \frac{\text{\# of trainable parameters in LETS-C}}{\text{\# of trainable parameters in OneFitsAll}}$ measures computational efficiency relative to OneFitsAll.

Accuracy ↑												
Method/Dataset		EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↑**	**AvgWins % ↑**
OneFitsAll		34.2	69.2	32.7	77.2	98.6	87.9	93.2	59.4	99.2	88.1	73.97	–
LETS-C	`text-embedding-3-large`	52.9	68.9	23.8	78	99.2	93.1	93.2	62.8	99.2	90.6	**76.17**	**80%**
	`e5-mistral-7b-instruct`	55.5	68.7	23.3	77.6	99.2	84.4	93.9	59.4	98.5	88.4	**74.89**	**60%**
	`gte-large-en-v1.5`	57.8	68.8	24.7	77.6	98.4	91.3	94.2	60	99	88.4	**76.02**	**60%**
	`nomic-embed-text-v1`	52.9	68	24.8	76.6	99.2	88.4	93.9	59.4	98.6	89.4	**75.12**	**60%**
Trainable Parameters (M) ↓												
Method/Dataset		EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↓**	
OneFitsAll		1.42	2.37	1.73	2.03	1.32	10.23	0.98	1.04	1.82	1.0	2.39	
LETS-C	`text-embedding-3-large`	0.28	0.003	0.15	0.04	0.14	0.56	0.30	0.33	0.14	0.26	**0.22**	
	Ratio %	19.89	0.16	8.89	2.28	11.19	5.51	30.83	32.06	7.77	26.21	**14.48**	
	`e5-mistral-7b-instruct`	0.13	0.40	0.39	0.17	0.16	0.30	0.24	0.24	0.33	0.16	**0.25**	
	Ratio %	9.48	16.93	22.78	8.54	12.55	2.97	25.06	23.59	18.39	15.93	**15.62**	
	`gte-large-en-v1.5`	0.18	0.31	0.31	0.12	0.04	0.56	0.03	0.07	0.22	0.09	**0.19**	
	Ratio %	12.91	13.44	18.27	6.11	3.36	5.51	3.77	7.65	12.34	9.00	**9.24**	
	`nomic-embed-text-v1`	0.03	0.03	0.36	0.07	0.05	0.19	0.02	0.10	0.06	0.02	**0.09**	
	Ratio %	2.21	1.31	20.99	3.58	3.99	1.85	3.02	10.03	3.34	2.78	**5.31**	

has the highest parameter count (0.25M) due to its largest dimensionality of 4096, followed by `text-embedding-3-large` (3072 dimensions, 0.22M), `gte-large-en-v1.5` (1024 dimensions, 0.19M), and `nomic-embed-text-v1` (768 dimensions, 0.09M). Consequently, our approach generalizes across diverse text embedding models, demonstrates superior performance, with the benefit of being lightweight.

L Trade-offs: Model Accuracy vs. Parameter Complexity

Table 11 illustrates the trade-off between model accuracy and the complexity of training parameters in our model, which utilizes both embeddings and time series data as inputs to a lightweight framework.

This trade-off between model accuracy and parameter complexity is data-dependent. However, we generally observe a trend where a reduction in parameters leads to only a slight decrease in accuracy. Next, let's take a closer look at three datasets: Heartbeat, PEMS-SF, and Spoken Arabic Digits, to understand how these trade-offs manifest in different contexts.

Heartbeat: In the Heartbeat dataset, we retain 99.48% of the optimal model's accuracy using only 75% of its trainable parameters. Specifically, the optimal model achieves an accuracy of 78% with 46,426 trainable parameters, while the second-best model achieves 77.6% accuracy with 34,820 parameters. Moreover, we retain 96.28% accuracy with a further reduction to 57.95% of the parameters.

PEMS-SF: In the PEMS-SF dataset, the optimal model starts with an accuracy of 93.1% and 564,231 trainable parameters. Reducing the parameters to 173,866 (30.81% of the original), the model maintains 98.06% of its optimal accuracy at 91.3%. Further parameter reductions to 85,210 (15.10%), 69,077 (12.24%), and 62,901 (11.14%) result in accuracies of 90.8%, 87.9%, and 87.3%, respectively. These reductions illustrate that even significant reductions in parameters only lead to a slight decrease in performance.

SpokenArabicDigits: For the Spoken Arabic Digits dataset, reducing the number of trainable parameters generally correlates with a minor decline in accuracy, though the trade-off is modest. The optimal model, achieving an accuracy of 99.2% with 141,790 trainable parameters, shows that even with substantial reductions to 70,066 parameters (49.41% of the original), the accuracy remains high at 99%, retaining 99.79% of the original model's accuracy. Further reductions to 46,658 (32.90%), 30,964 (21.83%), 20,646 (14.56%), 15,487 (10.92%), 10,328 (7.28%), and 5,308 (3.74%) yield accuracies of 98.4%, 98.1%, 98%, 97.8%, 97.6%, and 96.6% respectively.

These examples highlight that efficiency in terms of trainable parameters does not linearly correspond to a loss in model accuracy across various datasets. While fewer parameters generally lead to a lower accuracy, the decrement is often proportional and manageable, making these models highly suitable for deployment in resource-constrained environments or for applications requiring rapid processing with minimal computational overhead.

Table 11: Trade-off between model accuracy and the complexity of training parameters. The accuracy and parameters of the best model are highlighted in **bold**. The accuracy difference is calculated as the raw difference between the accuracies of the reduced model and the best model. The % Delta in accuracy and parameters is defined separately for each as $100 \times \frac{\text{Accuracy of the reduced model}}{\text{Accuracy of the optimal model}}$ and $100 \times \frac{\text{Parameters of the reduced model}}{\text{Parameters of the optimal model}}$, quantifying the accuracy and computational efficiency of the reduced model relative to our best model.

Dataset	Accuracy (%) ↑	Trainable Parameters ↓	Difference | % Delta in Accuracy ↑	% Delta in Parameters ↓
EthanolConcentration	**52.9**	**283950**	-	-
	46	105344	-6.9 | 86.95	37.09
FaceDetection	**68.9**	**3842**	-	-
	68.6	2402	-0.3 | 99.56	62.51
	67.9	962	-1.0 | 98.54	25.03
	66.4	482	-2.5 | 96.37	12.54
Handwriting	**23.8**	**154526**	-	-
	23.2	107226	-0.6 | 97.47	69.39
	22.7	53626	-1.1 | 95.37	34.70
	20.2	20394	-3.6 | 84.87	13.19
	19.4	13426	-4.4 | 81.51	8.68
Heartbeat	**78**	**46426**	-	-
	77.6	34820	-0.4 | 99.48	75.00
	75.1	26908	-2.9 | 96.28	57.95
Japanese Vowels	**99.2**	**148233**	-	-
	98.9	123401	-0.3 | 99.69	83.24
	98.6	105353	-0.6 | 99.39	71.07
	98.4	100857	-0.8 | 99.19	68.03
PEMS-SF	**93.1**	**564231**	-	-
	91.3	173866	-1.8 | 98.06	30.81
	90.8	85210	-2.3 | 97.52	15.10
	87.9	69077	-5.2 | 94.41	12.24
	87.3	62901	-5.8 | 93.77	11.14
Self-Regulation SCP1	**93.2**	**302626**	-	-
	92.5	99657	-0.7 | 99.24	32.93
Self-Regulation SCP2	**62.8**	**334402**	-	-
	58.9	166306	-3.9 | 93.78	49.73
	57.8	111106	-5.0 | 92.03	33.22
	57.2	83330	-5.6 | 91.08	24.91
	56.1	76386	-6.7 | 89.33	22.84
Spoken Arabic Digits	**99.2**	**141790**	-	-
	99	70066	-0.2 | 99.79	49.41
	98.4	46658	-0.8 | 99.19	32.90
	98.1	30964	-1.1 | 98.89	21.83
	98	20646	-1.2 | 98.79	14.56
	97.8	15487	-1.4 | 98.58	10.92
	97.6	10328	-1.6 | 98.38	7.28
	96.6	5308	-2.6 | 97.37	3.74
UWave Gesture Library	**90.6**	**263338**	-	-
	88.4	211556	-2.2 | 97.57	80.33
	87.5	176298	-3.1 | 96.57	66.94

M Figures and Tables at Higher Resolution from the Main Paper

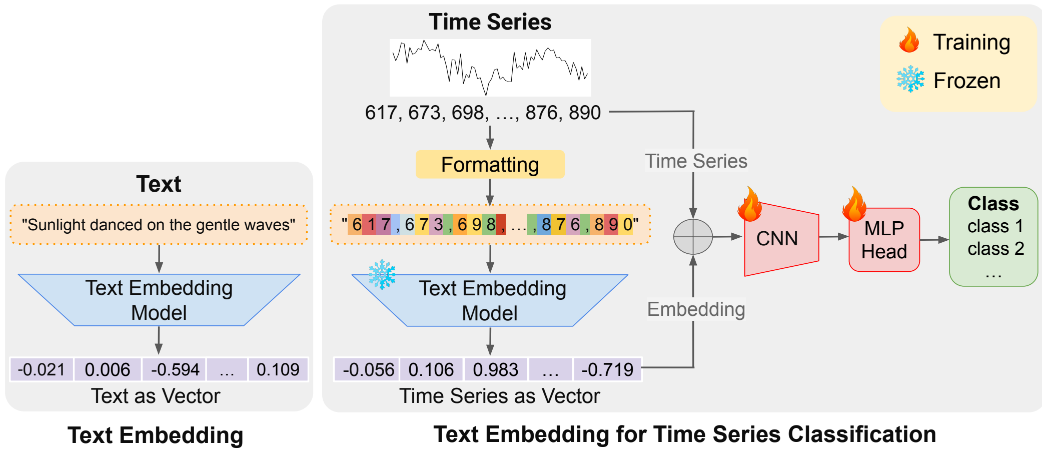

Figure 6: **Left:** Conventional text embedding. **Right:** Our proposed LETS-C framework normalizes time series, formats them for digit-specific tokenization, embeds them into an embedding space, and fuses the embeddings with the time series using element-wise addition. It uses a simple classification head with a CNN and an MLP. Only the CNN model and MLP head are trained in this framework.

Table 12: Comparison of classification accuracy (%) and AvgWins (%). **Red:** Best, Blue: Second best. **Abbreviations:** EC: Ethanol Concentration, FD: Face Detection, HW: Handwriting, HB: Heartbeat, JV: Japanese Vowels, PEMS: PEMS-SF, SCP1: Self-Regulation SCP1, SCP2: Self-Regulation SCP2, SAD: Spoken Arabic Digits, UW: UWave Gesture Library.

Model/Dataset		EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↑**	**AvgWins % ↑**
Classical methods	DTW	32.3	52.9	28.6	71.7	94.9	71.1	77.7	53.9	96.3	90.3	66.97	0%
	XGBoost	43.7	63.3	15.8	73.2	86.5	98.3	84.6	48.9	69.6	75.9	65.98	10%
	ROCKET	45.2	64.7	58.8	75.6	96.2	75.1	90.8	53.3	71.2	94.4	72.53	20%
MLP	LightTS	29.7	67.5	26.1	75.1	96.2	88.4	89.8	51.1	100	80.3	70.42	10%
	DLinear	32.6	68	27	75.1	96.2	75.1	87.3	50.5	81.4	82.1	67.53	0%
RNN	LSTM	32.3	57.7	15.2	72.2	79.7	39.9	68.9	46.6	31.9	41.2	48.56	0%
	LSTNet	39.9	65.7	25.8	77.1	98.1	86.7	84	52.8	100	87.8	71.79	10%
	LSSL	31.1	66.7	24.6	72.7	98.4	86.1	90.8	52.2	100	85.9	70.85	10%
CNN	TCN	28.9	52.8	53.3	75.6	98.9	68.8	84.6	55.6	95.6	88.4	70.25	0%
	TimesNet	35.7	68.6	32.1	78	98.4	89.6	91.8	57.2	99	85.3	73.57	0%
Transformers	Transformer	32.7	67.3	32	76.1	98.7	82.1	92.2	53.9	98.4	85.6	71.9	0%
	Reformer	31.9	68.6	27.4	77.1	97.8	82.7	90.4	56.7	97	85.6	71.52	0%
	Informer	31.6	67	32.8	80.5	98.9	81.5	90.1	53.3	100	85.6	72.13	20%
	Pyraformer	30.8	65.7	29.4	75.6	98.4	83.2	88.1	53.3	99.6	83.4	70.75	0%
	Autoformer	31.6	68.4	36.7	74.6	96.2	82.7	84	50.6	100	85.9	71.07	10%
	Stationformer	32.7	68	31.6	73.7	99.2	87.3	89.4	57.2	100	87.5	72.66	20%
	FEDformer	31.2	66	28	73.7	98.4	80.9	88.7	54.4	100	85.3	70.66	10%
	ETSformer	28.1	66.3	32.5	71.2	95.9	86	89.6	55	100	85	70.96	10%
	Flowformer	33.8	67.6	33.8	77.6	98.9	83.8	92.5	56.1	98.8	86.6	72.95	0%
LLM	OneFitsAll	34.2	69.2	32.7	77.2	98.6	87.9	93.2	59.4	99.2	88.1	73.97	20%
	LETS-C	52.9	68.9	23.8	78	99.2	93.1	93.2	62.8	99.2	90.6	**76.17**	**40%**

Table 13: Comparison of trainable parameters (millions) for LETS-C vs. OneFitsAll, with the Ratio $(\%) = 100 \times \frac{\text{\# of LETS-C parameters}}{\text{\# of OneFitsAll parameters}}$ illustrating LETS-C's efficiency relative to OneFitsAll.

Model/Dataset	EC	FD	HW	HB	JV	PEMS-SF	SCP1	SCP2	SAD	UW	**Average ↓**
LETS-C (Ours)	0.28	0.003	0.15	0.04	0.14	0.56	0.30	0.33	0.14	0.26	0.22
OneFitsAll	1.42	2.37	1.73	2.03	1.32	10.23	0.98	1.04	1.82	1.0	2.39
Ratio (%)	19.89	0.16	8.89	2.28	11.19	5.51	30.83	32.06	7.77	26.21	**14.48**

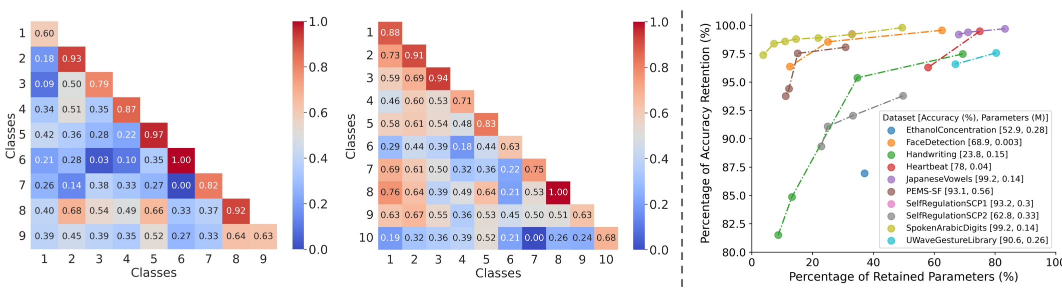

Figure 7: **Left:** Heatmaps illustrating within-class and between-class cosine similarities of text embeddings derived from the training time series data in JapaneseVowels **(far left)** with 9 classes and SpokenArabicDigits **(middle)** with 10 classes. **Right:** Trade-off between the percentage of accuracy retention and model parameter retention relative to LETS-C's optimal values across all datasets. The optimal LETS-C accuracy (%) and parameters (millions) for each dataset are detailed in the legend.