Self-Supervised Learning of Disentangled Representations for Multivariate Time-Series

Ching Chang, Chiao-Tung Chan, Wei-Yao Wang, Wen-Chih Peng, Tien-Fu Chen
National Yang Ming Chiao Tung University, Hsinchu, Taiwan
blacksnail789521.cs10@nycu.edu.tw, carol.chou.tun.ece05g@g2.nctu.edu.tw,
sf1638.cs05@nctu.edu.tw, {wcpeng, tfchen}@cs.nycu.edu.tw

Abstract

Multivariate time-series data in fields like healthcare and industry are informative but challenging due to high dimensionality and lack of labels. Recent self-supervised learning methods excel in learning rich representations without labels but struggle with disentangled embeddings and inductive bias issues like transformation-invariance. To address these challenges, we introduce TimeDRL, a framework for multivariate time-series representation learning with dual-level disentangled embeddings. TimeDRL features: (i) disentangled timestamp-level and instance-level embeddings using a [CLS] token strategy; (ii) timestamp-predictive and instance-contrastive tasks for representation learning; and (iii) avoidance of augmentation methods to eliminate inductive biases. Experiments on forecasting and classification datasets show TimeDRL outperforms existing methods, with further validation in semi-supervised settings with limited labeled data.

1 Introduction

Multivariate time-series data are critical in applications like power forecasting [1, 2] and smartwatch activity classification [3, 4], but they require extensive labeled data due to their complexity. To overcome this, researchers are increasingly using unsupervised representation learning, especially self-supervised learning (SSL), to extract embeddings from large unlabeled datasets, which can then be fine-tuned with limited labeled data for specific tasks. SSL has shown success in fields like NLP [5, 6] and computer vision [7, 8, 9], but its application to time-series data presents two challenges.

The first challenge is learning *disentangled dual-level representations*. Existing methods typically focus on either timestamp-level [10, 11] or instance-level embeddings [12, 13, 14], but not both, despite their distinct roles—timestamp-level for tasks like forecasting and anomaly detection, and instance-level for classification and clustering [15]. While instance-level embeddings can theoretically be derived from timestamp-level ones via pooling [10], this often leads to anisotropy issues, limiting their expressiveness [16, 17, 18]. The second challenge is *inductive bias*, which arises from inappropriate data augmentation methods borrowed from other domains, such as image rotation or masking in NLP [19, 20, 5], which can distort temporal patterns crucial to time-series analysis. Even time-series-specific augmentations like permutation [21] and cropping [10] may impose limiting assumptions about transformation invariance, overlooking the diverse nature of time-series datasets.

To address the challenges in time-series SSL, we propose **TimeDRL**, a framework that learns disentangled dual-level embeddings for multivariate time-series, optimizing both timestamp-level and instance-level representations for broad applicability across downstream tasks. The contributions are:

- **Disentangled Dual-Level Embeddings**: TimeDRL introduces a [CLS] token strategy with patched time-series data, enabling effective learning of both timestamp-level and instance-level embeddings, ensuring rich semantic capture for diverse tasks.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

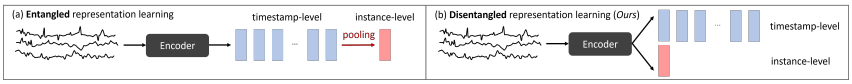

Figure 1: **Two categories of time-series representation learning.** (a) Entangled learning derives timestamp-level embeddings first, followed by pooling to extract instance-level embeddings. (b) Disentangled learning, as in TimeDRL, separately derives both embedding types.

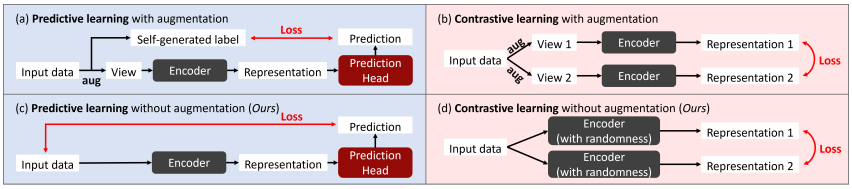

Figure 2: **Two categories of self-supervised learning.** (a) and (c) show predictive learning, using a single representation to predict inherent features. (b) and (d) depict contrastive learning, focusing on distinguishing data differences. TimeDRL avoids augmentation in both to prevent inductive bias.

- **Dual Pretext Tasks**: Two tailored pretext tasks are employed—timestamp-predictive for timestamp-level optimization and instance-contrastive for instance-level—ensuring specialized and effective learning for each level.
- **Mitigation of Inductive Bias**: TimeDRL avoids common inductive biases by not applying external augmentations, instead using dropout layers in the instance-contrastive task and a reconstruction approach without masking in the timestamp-predictive task.
- **Proven Effectiveness**: TimeDRL outperforms state-of-the-art methods across 11 real-world time-series forecasting and classification benchmarks, demonstrating its generalizability.

2 Method

Given an *unlabeled* set of N multivariate time-series samples $D_u = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$, the goal is to develop an encoder network f_θ that maps each sample $\mathbf{x}^{(n)}$ to its corresponding representation $\mathbf{z}^{(n)}$. For simplicity, the sample index (n) is omitted below. The encoder f_θ produces either (i) **Timestamp-Level Embedding**: $\mathbf{x} \in \mathbb{R}^{T \times C}$ is encoded into $\mathbf{z}_t \in \mathbb{R}^{T \times D_t}$, where T is the sequence length, C is the number of features, and D_t is the dimension of the timestamp-level embedding, or (ii) **Instance-Level Embedding**: $\mathbf{x} \in \mathbb{R}^{T \times C}$ is encoded into $\mathbf{z}_i \in \mathbb{R}^{D_i}$, where D_i is the dimension of the instance-level embedding. Here, $\mathbf{z}_i$ represents the overall information of the entire time-series $\mathbf{x}$.

2.1 Disentangled Dual-Level Embeddings

In BERT and RoBERTa, the [CLS] token is used for sentence-level embeddings, which inspired us to adopt it for instance-level embeddings in the time-series domain. Although instance-level embeddings can be derived from timestamp-level ones through pooling methods like global average pooling [10], this can lead to anisotropy problems [16, 17, 18], where embeddings are confined to a narrow region in the space, limiting their diversity. NLP studies [22, 23] have shown that optimizing the [CLS] token through contrastive learning produces better results than traditional pooling methods.

To generate embeddings, we normalize the input $\mathbf{x} \in \mathbb{R}^{T \times C}$ using instance normalization (IN) [24] and apply patching to produce $\mathbf{x}_{patched} \in \mathbb{R}^{T_p \times C \cdot P}$, where T_p is the number of patches and P the patch length. Later, a [CLS] token is appended, forming the encoder input $\mathbf{x}_{enc\_in} \in \mathbb{R}^{(1+T_p) \times C \cdot P}$. The encoder f_θ consists of a linear token encoding layer $W_{token} \in \mathbb{R}^{D \times C \cdot P}$, a positional encoding layer $PE \in \mathbb{R}^{(1+T_p) \times D}$, and Transformer blocks TBs. This produces the final embeddings $\mathbf{z} \in \mathbb{R}^{(1+T_p) \times D}$ as $\mathbf{z} = \text{TBs}(\mathbf{x}_{enc\_in} W_{token}^{\top} + PE)$. The instance-level embedding $\mathbf{z}_i \in \mathbb{R}^{D}$ is extracted from the first position of the embedding $\mathbf{z}[0, :]$, corresponding to the [CLS] token, while the timestamp-level embedding $\mathbf{z}_t \in \mathbb{R}^{T_p \times D}$ is derived from the remaining positions $\mathbf{z}[1 : T_p + 1, :]$. This approach avoids the anisotropy problem and leverages the strengths of Transformers in time-series SSL.

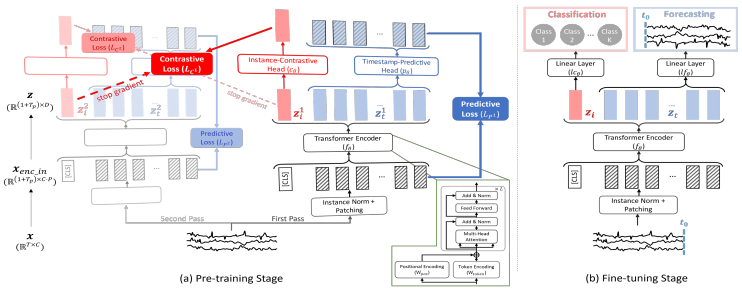

Figure 3: **TimeDRL Framework**. The framework has two stages: (a) *Pre-training* and (b) *Fine-tuning*. In (a), a Siamese network with a Transformer encoder generates two views of embeddings using dropout layer randomness, avoiding data augmentations. In (b), these embeddings are fine-tuned for time-series forecasting and classification tasks, highlighting TimeDRL's adaptability.

2.2 Timestamp-Predictive Task in TimeDRL

To capture relationships between timestamps, we develop a timestamp-predictive task that derives timestamp-level embeddings through predictive loss, avoiding inductive bias. Unlike methods in NLP [5, 25] and time-series [26] that rely on augmentation like masked language modeling (MLM), TimeDRL focuses on direct reconstruction of patched time-series data without augmentation.

Given a timestamp-level embedding $\mathbf{z}_t$, it passes through a predictive head p_θ (a linear layer) to generate a prediction. The predictive loss $\mathcal{L}_P$ is calculated as the Mean Squared Error (MSE) between the original patched data $\mathbf{x}_{patched}$ and the predicted output: $\mathcal{L}_P = \text{MSE}(\mathbf{x}_{patched}, p_\theta(\mathbf{z}_t))$. The instance-level embeddings $\mathbf{z}_i$ are not updated by this loss. Since the input $\mathbf{x}$ is processed twice to generate two views, $\mathbf{z}^1$ and $\mathbf{z}^2$, the predictive loss is computed for both timestamp-level embeddings $\mathbf{z}_t^1$ and $\mathbf{z}_t^2$ as: $\mathcal{L}_{P^1} = \text{MSE}(\mathbf{x}_{patched}, p_\theta(\mathbf{z}_t^1))$ and $\mathcal{L}_{P^2} = \text{MSE}(\mathbf{x}_{patched}, p_\theta(\mathbf{z}_t^2))$. The total predictive loss $\mathcal{L}_P$ is the average of $\mathcal{L}_{P^1}$ and $\mathcal{L}_{P^2}$: $\mathcal{L}_P = \frac{1}{2}\mathcal{L}_{P^1} + \frac{1}{2}\mathcal{L}_{P^2}$.

2.3 Instance-Contrastive Task in TimeDRL

To capture the overall information of the entire series, we develop an instance-contrastive task to derive instance-level embeddings through contrastive loss. In contrastive learning, two different views of embeddings are needed. To avoid data augmentations, we introduce randomness using dropout layers within the encoder, generating two distinct views by passing the data through the encoder twice: $\mathbf{z}^1 = f_\theta(\mathbf{x}_{patched})$ and $\mathbf{z}^2 = f_\theta(\mathbf{x}_{patched})$. The first position of each embedding, $\mathbf{z}_i^1 = \mathbf{z}^1[0,:]$ and $\mathbf{z}_i^2 = \mathbf{z}^2[0,:]$, is used as the instance-level embedding, avoiding external data augmentations and inductive bias. To address sampling bias in contrastive learning, we remove negative samples and focus exclusively on positive pairs, preventing model collapse by using a stop-gradient operation.

After obtaining instance-level embeddings $\mathbf{z}_i^1$ and $\mathbf{z}_i^2$, they are passed through an instance-contrastive head c_θ (a two-layer MLP) to produce $\hat{\mathbf{z}}_i^1 = c_\theta(\mathbf{z}_i^1)$ and $\hat{\mathbf{z}}_i^2 = c_\theta(\mathbf{z}_i^2)$. The contrastive loss is calculated to align $\hat{\mathbf{z}}_i^1$ with $\mathbf{z}_i^2$ using negative cosine similarity: $\mathcal{L}_{C^1} = -\text{cosine}(\hat{\mathbf{z}}_i^1, \text{stop\_gradient}(\mathbf{z}_i^2))$. A symmetric loss is also calculated: $\mathcal{L}_{C^2} = -\text{cosine}(\hat{\mathbf{z}}_i^2, \text{stop\_gradient}(\mathbf{z}_i^1))$. The total contrastive loss $\mathcal{L}_C$ is the average of $\mathcal{L}_{C^1}$ and $\mathcal{L}_{C^2}$: $\mathcal{L}_C = \frac{1}{2}\mathcal{L}_{C^1} + \frac{1}{2}\mathcal{L}_{C^2}$. Finally, the overall loss combines timestamp-predictive and instance-contrastive tasks, with λ balancing them: $\mathcal{L} = \mathcal{L}_P + \lambda \cdot \mathcal{L}_C$.

3 Experiments

We evaluate TimeDRL in two key areas: *forecasting*, testing timestamp-level embeddings, and *classification*, focusing on instance-level embeddings.

3.1 Linear Evaluation on Time-Series Forecasting (TSF)

To evaluate TimeDRL's timestamp-level embeddings, we perform a linear evaluation on time-series forecasting. The encoder is pre-trained with pretext tasks, then frozen, and a linear layer is trained for the downstream forecasting task. Following SimTS [11], we set prediction lengths $T \in$

Table 1: **Linear Eval on Multivariate TSF.**

Methods	Unsupervised Representation Learning										End-to-end Forecasting			
	TimeDRL		SimTS		TS2Vec		TNC		CoST		Informer		TCN	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	**0.396**	**0.421**	0.643	0.582	0.794	0.650	0.904	0.702	0.650	0.585	0.907	0.934	0.872	0.692
ETTh2	**0.303**	**0.360**	1.165	0.798	1.544	0.947	1.869	1.053	1.283	0.851	2.371	1.199	2.564	1.216
ETTm1	**0.321**	**0.364**	0.393	0.423	0.631	0.554	0.740	0.599	0.419	0.439	0.749	0.640	0.751	0.610
ETTm2	**0.208**	**0.283**	0.572	0.502	0.652	0.543	0.784	0.608	0.644	0.534	1.003	0.654	2.042	0.960
Exchange	**0.247**	**0.315**	0.789	0.613	0.835	0.622	0.732	0.583	1.049	0.742	1.308	0.888	2.689	1.356
Weather	**0.199**	**0.239**	0.424	0.458	0.456	0.476	0.445	0.470	0.430	0.464	0.574	0.552	0.440	0.461

Table 2: **Linear Eval on Univariate TSF.**

Methods	Unsupervised Representation Learning										End-to-end Forecasting			
	TimeDRL		SimTS		TS2Vec		TNC		CoST		Informer		TCN	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	**0.061**	**0.188**	0.080	0.210	0.116	0.258	0.147	0.301	0.091	0.228	0.186	0.327	0.326	0.470
ETTh2	**0.147**	**0.297**	0.159	0.306	0.166	0.319	0.168	0.322	0.161	0.307	0.204	0.358	0.180	0.335
ETTm1	**0.036**	**0.138**	0.059	0.158	0.065	0.181	0.079	0.205	0.054	0.164	0.194	0.337	0.134	0.270
ETTm2	**0.085**	**0.205**	0.109	0.209	0.112	0.243	0.114	0.248	0.098	0.221	0.136	0.273	0.134	0.268
Exchange	**0.346**	**0.375**	0.443	0.423	0.419	0.427	0.524	0.502	0.455	0.431	0.908	0.646	0.390	0.407
Weather	**0.043**	**0.127**	0.162	0.303	0.181	0.308	0.175	0.303	0.183	0.307	0.223	0.370	0.166	0.291

$\{24, 48, 168, 336, 720\}$ for ETTh1, ETTh2, Exchange, and Weather, and $T \in \{24, 48, 96, 228, 672\}$ for ETTm1 and ETTm2. TimeDRL demonstrates its versatility with an average MSE improvement of 58.02% for multivariate and 29.09% for univariate forecasting (Table 1 and 2).

3.2 Linear Evaluation on Time-Series Classification (TSC)

To assess TimeDRL's instance-level embeddings, we use a linear evaluation for time-series classification. TimeDRL shows an average accuracy improvement of 1.48% over state-of-the-art methods (Table 3). On challenging datasets like FingerMovements, TimeDRL achieves a 22.86% accuracy boost and a 58.13% improvement in Cohen's Kappa. In the Epilepsy dataset, TimeDRL's accuracy is only 0.07% lower than the best baseline, demonstrating its strong performance with univariate data.

Table 3: **Linear evaluation on TSC.**

Dataset	Metric	TimeDRL	MHCCL	CCL	SimCLR	BYOL	TS2Vec	TSTCC	T-Loss
FingerMovements	ACC	**64.00**	52.09	50.23	49.20	49.60	50.00	50.17	50.50
	MF1	**63.77**	50.51	47.32	43.08	49.38	49.99	49.07	50.33
	κ	**28.26**	17.87	9.18	-0.69	-0.49	-0.01	0.33	4.01
PenDigits	ACC	98.00	**98.69**	91.27	89.24	94.93	97.83	97.44	97.86
	MF1	98.01	**98.71**	88.61	89.17	94.96	97.80	97.45	97.87
	κ	**97.78**	97.43	87.66	88.04	94.37	97.59	97.16	97.63
HAR	ACC	89.01	**91.60**	86.84	81.06	89.46	90.47	89.22	91.06
	MF1	89.41	**91.77**	83.56	80.62	89.31	90.46	89.23	90.94
	κ	86.79	**89.90**	81.46	77.25	87.33	89.15	87.03	89.26
Epilepsy	ACC	97.78	97.85	95.47	93.00	**98.08**	96.32	97.19	96.94
	MF1	96.53	95.44	91.38	88.09	**96.99**	94.27	95.47	95.20
	κ	93.07	91.08	79.42	76.27	**93.99**	88.54	90.94	90.41
WISDM	ACC	91.45	**93.60**	85.18	83.04	87.84	92.33	81.48	91.48
	MF1	82.37	**91.70**	81.22	75.83	84.02	90.27	69.17	88.79
	κ	87.85	**90.96**	79.19	75.15	82.43	90.36	73.13	87.79

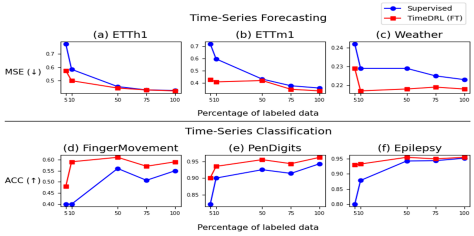

Figure 4: **Semi-supervised learning.**

3.3 Semi-supervised learning

Self-supervised learning thrives in semi-supervised settings with limited labeled data and abundant unlabeled data. We first pre-train an encoder on unlabeled data, then fine-tune it with limited labeled data, adjusting the encoder weights during fine-tuning. To simulate limited labels, we withhold some labels in our datasets. Figure 4 shows that TimeDRL boosts forecasting and classification performance, especially as labeled data decreases, with benefits evident even with full label availability.

3.4 Ablation Study

We conducted ablation studies to assess the impact of key components in the TimeDRL framework, including pretext tasks, data augmentation, pooling methods, encoder architectures, and stop-gradient operations. The results highlight the importance of each component in enhancing the model's performance across different time-series tasks. Detailed findings are provided in Appendix B.8.

4 Conclusion

This paper presents TimeDRL, a novel framework for multivariate time-series representation learning with disentangled dual-level embeddings, optimized for tasks like forecasting and classification. TimeDRL uses a [CLS] token strategy to extract instance-level embeddings and employs two pretext tasks: a timestamp-predictive task for timestamp-level embeddings and an instance-contrastive task for instance-level embeddings. To prevent inductive bias, TimeDRL avoids direct data augmentations, relying instead on reconstruction error and dropout randomness. Experiments on 6 forecasting and 5 classification datasets show TimeDRL's superiority, with a 58.02% improvement in MSE for forecasting and a 1.48% accuracy boost for classification. TimeDRL also excels in semi-supervised learning with limited labeled data. Future work will enhance TimeDRL for classification and explore comparisons with large language models (LLMs) in time-series analysis.

References

[1] 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, pages 11106–11115, 2021.

[2] Han Wang, Ning Zhang, Ershun Du, Jie Yan, Shuang Han, and Yongqian Liu. A comprehensive review for wind, solar, and electrical load forecasting methods. *Global Energy Interconnection*, 5(1):9–30, 2022.

[3] Gökhan Şengül, Murat Karakaya, Sanjay Misra, Olusola O Abayomi-Alli, and Robertas Damaševičius. Deep learning based fall detection using smartwatches for healthcare applications. *Biomedical Signal Processing and Control*, 71:103242, 2022.

[4] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[5] 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.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[8] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[9] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[10] 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, pages 8980–8987, 2022.

[11] Xiaochen Zheng, Xingyu Chen, Manuel Schürch, Amina Mollaysa, Ahmed Allam, and Michael Krauthammer. Simts: Rethinking contrastive representation learning for time series forecasting. *arXiv preprint arXiv:2303.18205*, 2023.

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

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

[14] Qianwen Meng, Hangwei Qian, Yong Liu, Lizhen Cui, Yonghui Xu, and Zhiqi Shen. Mhccl: masked hierarchical cluster-wise contrastive learning for multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9153–9161, 2023.

[15] Kexin Zhang, Qingsong Wen, Chaoli Zhang, Rongyao Cai, Ming Jin, Yong Liu, James Zhang, Yuxuan Liang, Guansong Pang, Dongjin Song, et al. Self-supervised learning for time series analysis: Taxonomy, progress, and prospects. *arXiv preprint arXiv:2306.10125*, 2023.

[16] Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. Representation degeneration problem in training natural language generation models. *arXiv preprint arXiv:1907.12009*, 2019.

[17] Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. *arXiv preprint arXiv:1909.00512*, 2019.

[18] Lingxiao Wang, Jing Huang, Kevin Huang, Ziniu Hu, Guangtao Wang, and Quanquan Gu. Improving neural language generation with spectrum control. In *International Conference on Learning Representations*, 2019.

[19] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[20] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[21] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-series representation learning via temporal and contextual contrasting. *arXiv preprint arXiv:2106.14112*, 2021.

[22] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

[23] Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljačić, Shang-Wen Li, Wen-tau Yih, Yoon Kim, and James Glass. Diffcse: Difference-based contrastive learning for sentence embeddings. *arXiv preprint arXiv:2204.10298*, 2022.

[24] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

[25] 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*, 2019.

[26] 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.

[27] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[28] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[29] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in Neural Information Processing Systems*, 33:11033–11043, 2020.

[30] Dara Bahri, Heinrich Jiang, Yi Tay, and Donald Metzler. Scarf: Self-supervised contrastive learning using random feature corruption. *arXiv preprint arXiv:2106.15147*, 2021.

[31] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

[32] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. *arXiv preprint arXiv:2102.06514*, 2021.

[33] Edward Ma. Nlp augmentation. https://github.com/makcedward/nlpaug, 2019.

[34] Soma Onishi and Shoya Meguro. Rethinking data augmentation for tabular data in deep learning. *arXiv preprint arXiv:2305.10308*, 2023.

[35] Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z Li. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *Proceedings of the ACM Web Conference 2022*, pages 1070–1079, 2022.

[36] Xiang Zhang, Ziyuan Zhao, Theodoros Tsiligkaridis, and Marinka Zitnik. Self-supervised contrastive pre-training for time series via time-frequency consistency. *arXiv preprint arXiv:2206.08496*, 2022.

[37] 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*, pages 95–104, 2018.

[38] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra Perez, and Jorge Luis Reyes Ortiz. A public domain dataset for human activity recognition using smartphones. In *Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning*, pages 437–442, 2013.

[39] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907, 2001.

[40] Fevzi Alimoğlu and Ethem Alpaydin. Combining multiple representations for pen-based handwritten digit recognition. *Turkish Journal of Electrical Engineering and Computer Sciences*, 9(1):1–12, 2001.

[41] Benjamin Blankertz, Gabriel Curio, and Klaus-Robert Müller. Classifying single trial eeg: Towards brain computer interfacing. *Advances in neural information processing systems*, 14, 2001.

[42] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Cost: Contrastive learning of disentangled seasonal-trend representations for time series forecasting. *arXiv preprint arXiv:2202.01575*, 2022.

[43] 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.

[44] Vivek Sharma, Makarand Tapaswi, M Saquib Sarfraz, and Rainer Stiefelhagen. Clustering based contrastive learning for improving face representations. In *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*, pages 109–116. IEEE, 2020.

[45] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[46] Terry T Um, Franz MJ Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In *Proceedings of the 19th ACM international conference on multimodal interaction*, pages 216–220, 2017.

[47] 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*, pages 27268–27286. PMLR, 2022.

[48] Chaoning Zhang, Kang Zhang, Chenshuang Zhang, Trung X Pham, Chang D Yoo, and In So Kweon. How does simsiam avoid collapse without negative samples? a unified understanding with self-supervised contrastive learning. *arXiv preprint arXiv:2203.16262*, 2022.

A Related Work

A.1 Foundational Concepts of Self-Supervised Learning: A Pre-Time-Series Perspective

Self-Supervised Learning (SSL) techniques have proven to be effective for learning generic representations by designing pretext tasks, which are generally categorized into two main types: *predictive* and *contrastive* learning [27]. As shown in Figure 2, predictive learning involves using a single representation to forecast characteristics that are intrinsic to the data. Conversely, contrastive learning focuses on discerning subtle differences between data samples by calculating loss based on pairs of representations. Siamese networks [28], which are weight-sharing neural networks, are frequently utilized in contrastive learning to process input pairs simultaneously.

The concepts of predictive and contrastive learning initially arose in the fields of Natural Language Processing (NLP) and Computer Vision (CV). In NLP, BERT [5] introduces predictive tasks such as masked language modeling and next sentence prediction to derive semantically rich representations, whereas GPT [6] uses autoregressive predictive tasks to showcase its few-shot learning capabilities. SimCSE [22] enhances sentence-level embeddings using contrastive tasks applied to the [CLS] token, incorporating dropout layers to add variability without the need for external augmentation techniques. In CV, SimCLR [7] leverages contrastive learning to create detailed representations by treating different augmented views of the same instance as positive pairs, with all other instances in the minibatch treated as negative pairs. BYOL [8] and SimSiam [9], however, implement an additional prediction head along with a stop-gradient strategy to circumvent the use of negative samples and to eliminate the requirement for large batch sizes.

Recently, SSL methods have been adopted in new domains, such as tabular data and Graph Neural Networks (GNNs). In the realm of tabular data, VIME [29] introduces a predictive task involving the estimation of a mask vector using an autoencoder structure, while SCARF [30] applies contrastive learning to capture more refined representations. In GNNs, GraphCL [31] and BGRL [32] have successfully implemented contrastive learning in tasks such as graph classification and edge prediction. However, applying SSL techniques across domains often introduces inductive bias. For example, data augmentation methods used in CV, such as image colorization [19] and rotation [20], or masking [5] and synonym replacement [33] in NLP, may lead to biases that are not suitable for the target domain. To mitigate this, domain-specific solutions have been proposed in various studies. For instance, MTR [34] in tabular data introduces a specially designed augmentation method for tabular formats. In GNNs, SimGRACE [35] completely avoids the use of data augmentation. Taking inspiration from this, TimeDRL eliminates the use of data augmentation across all pretext tasks to minimize any potential inductive biases.

A.2 Self-Supervised Learning for Time-Series Data

Self-supervised learning for time-series data representation has gained significant momentum in recent years. T-Loss [13] applies a triplet loss with time-based negative sampling to learn effective representations for time-series data. TNC [12] leverages the Augmented Dickey-Fuller (ADF) statistical test to identify temporal neighborhoods and employs Positive-Unlabeled (PU) learning to mitigate sampling bias. TS-TCC [21] generates two views of the data using strong and weak augmentations, then learns representations by contrasting temporal and contextual information across views. TS2Vec [10] focuses on capturing multi-scale contextual information at both the instance and timestamp levels, making it the first versatile framework applicable across various time-series tasks. TF-C [36] proposes encoding temporal neighborhoods to align closely with their frequency-based counterparts through time-frequency consistency. MHCCL [14] utilizes semantic information from a hierarchical structure of latent partitions in multivariate time-series, which is further refined by hierarchical clustering for improved positive and negative pair selection. SimTS [11] offers a streamlined approach to time-series forecasting by learning to predict future states from past data in a latent space, without relying on negative pairs or making specific assumptions about the time-series. Many self-supervised learning efforts for time-series data have primarily concentrated on generating instance-level embeddings by extracting them from timestamp-level embeddings via pooling methods [10]. However, this method often leads to an anisotropy issue, where embeddings become confined to a limited area within the embedding space, thereby restricting their expressiveness [16, 17, 18]. TimeDRL addresses this limitation by disentangling timestamp-level and instance-level embeddings to enhance their flexibility and representational power.

Table 4: **Statistical overview of the 7 datasets for time-series forecasting.**

Datasets	Features	Timesteps	Granularity
ETTh1 & ETTh2	7	17,420	1 hour
ETTm1 & ETTm2	7	69,680	5 min
Exchange	8	7,588	1 day
Weather	21	52,696	10 min

Table 5: **Statistical overview of the 5 datasets for time-series classification.**

Datasets	Samples	Features	Classes	Length
HAR	10,299	9	6	128
WISDM	4,091	3	6	256
Epilepsy	11,500	1	2	178
PenDigits	10,992	2	10	8
FingerMovements	416	28	2	50

B More on Experiments

B.1 Datasets

Datasets for Time-Series Forecasting For our time-series forecasting experiments, we used six real-world, publicly available benchmark datasets. Table 4 presents an overview of the key characteristics of each dataset, including the number of features, total dataset length, and sampling frequency.

ETT [1] contains long-term electric power deployment data. The dataset includes two hourly-sampled datasets (ETTh1, ETTh2) and two 15-minute-sampled datasets (ETTm1, ETTm2), covering two years from different provinces in China. It comprises one oil temperature feature and six power load features. For multivariate forecasting, all features are used, while only the oil temperature feature is employed for univariate forecasting. **Exchange** [37] includes daily exchange rates of eight foreign countries, spanning from 1990 to 2016. The countries include Australia, Britain, Canada, Switzerland, China, Japan, New Zealand, and Singapore. For multivariate forecasting, data from all countries are used, and for univariate forecasting, the focus is on Singapore's exchange rate. **Weather**[1] provides local climatological data collected from nearly 1,600 U.S. locations over four years. Each record contains 11 weather variables, with the 'web bulb' feature as the target. All variables are used for multivariate forecasting, while the 'web bulb' feature is utilized for univariate forecasting.

Datasets for Time-Series Classification For our time-series classification experiments, we used five real-world, publicly available benchmark datasets. Table 4 outlines the characteristics of each dataset, including the number of time-series samples, features, classes, and sample lengths.

HAR [38] includes sensor data from 30 subjects performing six activities. The data were collected using a Samsung Galaxy S2 device, and the task is to predict the activity based on accelerometer and gyroscope measurements. **WISDM** [4] comprises time-series data from accelerometers and gyroscopes in smartphones and smartwatches. The data were gathered from 51 participants performing 18 activities, with each activity recorded for three minutes. **Epilepsy** [39] contains EEG recordings from 500 individuals using a single-channel EEG sensor at a frequency of 174 Hz. Each subject's brain activity was recorded for 23.6 seconds, and the data were classified as either indicating epilepsy or not. **PenDigits** [40] focuses on handwritten digit classification, where 44 participants wrote digits from 0 to 9, with the x and y coordinates recorded. The data were captured at a 500x500 pixel resolution and resampled to eight spatial points. **FingerMovements** [41] consists of time-series data from subjects performing self-paced key typing on a computer keyboard. The task involved three six-minute sessions conducted on the same day, with breaks in between, and the subjects typed at an average speed of one key per second.

[1] https://www.ncei.noaa.gov/data/local-climatological-data/

B.2 Evaluation Metrics

Evaluation Metrics for Time-Series Forecasting In time-series forecasting, we mainly rely on Mean Squared Error (MSE) and Mean Absolute Error (MAE) as the primary evaluation metrics. The Mean Squared Error (MSE) is calculated as:

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{y}^{(n)} - \hat{\mathbf{y}}^{(n)})^2. \tag{1}$$

Here, $\mathbf{y}^{(n)}$ represents the actual future value corresponding to input $\mathbf{x}^{(n)}$, $\hat{\mathbf{y}}^{(n)}$ is the predicted value, and N is the total number of samples. The Mean Absolute Error (MAE) is defined as:

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}|\mathbf{y}^{(n)} - \hat{\mathbf{y}}^{(n)}|. \tag{2}$$

Evaluation Metrics for Time-Series Classification In time-series classification, the metrics we use include accuracy (ACC), macro-averaged F1-score (MF1), and Cohen's Kappa coefficient (κ). Accuracy is defined by:

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{3}$$

where TP, TN, FP, and FN stand for true positive, true negative, false positive, and false negative, respectively. The macro-averaged F1-score is determined by:

$$\text{MF1} = \frac{2 \times \text{P} \times \text{R}}{\text{P} + \text{R}}, \tag{4}$$

where Precision (P) is given by:

$$\text{P} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5}$$

and Recall (R) is defined as:

$$\text{R} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{6}$$

Cohen's Kappa coefficient is calculated as:

$$\kappa = \frac{\text{ACC} - p_e}{1 - p_e}, \tag{7}$$

where p_e is the expected probability of chance agreement, computed as:

$$p_e = \frac{(\text{TP} + \text{FN}) \times (\text{TP} + \text{FP}) + (\text{FP} + \text{TN}) \times (\text{FN} + \text{TN})}{N^2}, \tag{8}$$

with N representing the total number of samples. The Kappa coefficient ranges from -1 (complete disagreement) to 1 (perfect agreement), with 0 indicating no agreement beyond chance. Cohen's Kappa (κ) is particularly useful for evaluating classifiers on imbalanced datasets, as it adjusts for the probability of agreement by chance. This helps identify when classifier performance is comparable to random guessing (κ close to 0) or when κ is negative, indicating worse-than-random performance, offering valuable insights into addressing class imbalance.

B.3 Baselines

Baselines for Time-Series Forecasting **SimTS** [11] simplifies time-series forecasting by learning to predict future outcomes from past data within a latent space, avoiding the need for negative pairs or any specific assumptions about time-series properties. **TS2Vec** [10] is the first universal framework for time-series representation learning, emphasizing the differentiation of multi-scale contextual information at both instance and timestamp levels, and has shown effectiveness across a variety of time-series tasks. **TNC** [12] utilizes the Augmented Dickey-Fuller test to detect temporal neighborhoods, and applies Positive-Unlabeled learning to reduce the impact of sampling bias. **CoST** [42] combines contrastive losses from both time and frequency domains, allowing it to capture distinct trend and seasonal representations.

Table 6: **Linear evaluation on multivariate time-series forecasting.** We use prediction lengths $T \in \{24, 48, 168, 336, 720\}$ for ETTh1, ETTh2, Exchange, and Weather; and $T \in \{24, 48, 96, 228, 672\}$ for ETTm1 and ETTm2. The best results are in **bold**, while the second-best are underlined.

Methods		Unsupervised Representation Learning										End-to-end Forecasting			
		TimeDRL		SimTS		TS2Vec		TNC		CoST		Informer		TCN	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	**0.327**	**0.378**	0.377	0.422	0.590	0.531	0.708	0.592	0.386	0.429	0.577	0.549	0.583	0.547
	48	**0.353**	**0.392**	0.427	0.454	0.624	0.555	0.749	0.619	0.437	0.464	0.685	0.625	0.670	0.606
	168	**0.418**	**0.427**	0.638	0.577	0.762	0.639	0.884	0.699	0.643	0.582	0.931	0.752	0.811	0.680
	336	**0.437**	**0.446**	0.815	0.685	0.931	0.728	1.020	0.768	0.812	0.679	1.128	0.873	1.132	0.815
	720	**0.446**	**0.461**	0.956	0.771	1.063	0.799	1.157	0.830	0.970	0.771	1.215	1.869	1.165	0.813
ETTh2	24	**0.183**	**0.279**	0.336	0.434	0.424	0.489	0.612	0.595	0.447	0.502	0.720	0.665	0.935	0.754
	48	**0.229**	**0.308**	0.564	0.571	0.619	0.605	0.840	0.716	0.699	0.637	1.457	1.001	1.300	0.911
	168	**0.334**	**0.376**	1.407	0.926	1.845	1.074	2.359	1.213	1.549	0.982	3.489	1.515	4.017	1.579
	336	**0.372**	**0.407**	1.640	0.996	2.194	1.197	2.782	1.349	1.749	1.042	2.723	1.340	3.460	1.456
	720	**0.395**	**0.428**	1.878	1.065	2.636	1.370	2.753	1.394	1.971	1.092	3.467	1.473	3.106	1.381
ETTm1	24	**0.217**	**0.299**	0.232	0.314	0.453	0.444	0.522	0.472	0.246	0.329	0.323	0.369	0.522	0.472
	48	**0.279**	**0.339**	0.311	0.368	0.592	0.521	0.695	0.567	0.381	0.386	0.494	0.503	0.542	0.508
	96	**0.302**	**0.357**	0.360	0.402	0.635	0.554	0.731	0.595	0.378	0.419	0.678	0.614	0.666	0.578
	288	**0.377**	**0.398**	0.450	0.467	0.693	0.597	0.818	0.649	0.472	0.486	1.056	0.786	0.991	0.735
	672	**0.429**	**0.424**	0.612	0.563	0.782	0.653	0.932	0.712	0.620	0.574	1.192	0.926	1.032	0.756
ETTm2	24	**0.104**	**0.205**	0.108	0.223	0.180	0.293	0.185	0.297	0.122	0.244	0.173	0.301	0.180	0.324
	48	**0.138**	**0.236**	0.164	0.285	0.244	0.350	0.264	0.360	0.183	0.305	0.303	0.409	0.204	0.327
	96	**0.174**	**0.265**	0.271	0.376	0.360	0.427	0.389	0.458	0.294	0.394	0.365	0.453	3.041	1.330
	288	**0.270**	**0.326**	0.716	0.646	0.723	0.639	0.920	0.788	0.723	0.652	1.047	0.804	3.162	1.337
	672	**0.354**	**0.381**	1.600	0.979	1.753	1.007	2.164	1.135	1.899	1.073	3.126	1.302	3.624	1.484
Exchange	24	**0.026**	**0.110**	0.059	0.172	0.108	0.252	0.105	0.236	0.136	0.291	0.611	0.626	2.483	1.327
	48	**0.042**	**0.143**	0.135	0.265	0.200	0.341	0.162	0.270	0.250	0.387	0.680	0.644	2.328	1.256
	168	**0.146**	**0.279**	0.713	0.635	0.412	0.492	0.397	0.480	0.924	0.762	1.097	0.825	2.372	1.279
	336	**0.340**	**0.422**	1.409	0.938	1.339	0.901	1.008	0.866	1.774	1.063	1.672	1.036	3.113	1.459
	720	**0.679**	**0.620**	1.628	1.056	2.114	1.125	1.989	1.063	2.160	1.209	2.478	1.310	3.150	1.458
Weather	24	**0.101**	**0.145**	0.298	0.359	0.308	0.364	0.320	0.373	0.298	0.360	0.335	0.381	0.321	0.367
	48	**0.128**	**0.181**	0.359	0.410	0.375	0.417	0.380	0.421	0.359	0.411	0.395	0.459	0.386	0.423
	168	**0.194**	**0.244**	0.426	0.461	0.496	0.506	0.479	0.495	0.464	0.491	0.608	0.567	0.491	0.501
	336	**0.249**	**0.285**	0.504	0.520	0.532	0.533	0.505	0.514	0.497	0.517	0.702	0.620	0.502	0.507
	720	**0.323**	**0.341**	0.535	0.542	0.567	0.558	0.543	0.547	0.533	0.542	0.831	0.731	0.498	0.508

In addition to unsupervised representation learning methods, we also include two end-to-end learning approaches where representation learning and forecasting are trained together. **Informer** [1] tackles the challenges of quadratic time complexity and memory usage in the standard Transformer by introducing ProbSparse self-attention and distilling operations. **TCN** [43] integrates dilations and residual connections with causal convolutions, which are crucial for autoregressive forecasting.

Baselines for Time-Series Classification **MHCCL** [14] leverages semantic information from a hierarchical structure in multivariate time-series, using hierarchical clustering to enhance positive and negative sample pairing. **CCL** [44] uses a clustering-based approach for representation learning, generating discriminative features by utilizing labels obtained from clustering and constraints. **SimCLR** [7] applies contrastive learning, treating augmented views of the same instance as positive pairs and different instances within the minibatch as negatives. **BYOL** [8] employs two networks—online and target—that interact, where the online network predicts the target's representation under varying augmentations, and the target network is updated as a slow-moving average of the online network. **TS2Vec** [10], known as a universal framework for time-series representation learning, is also used in our analysis for time-series classification. **TS-TCC** [21] creates two views using strong and weak augmentations, and learns representations by contrasting these views in both temporal and contextual dimensions. **T-Loss** [13] learns representations by training with triplet loss, using time-based negative sampling.

Table 7: **Linear evaluation on univariate time-series forecasting.** We use prediction lengths $T \in \{24, 48, 168, 336, 720\}$ for ETTh1, ETTh2, Exchange, and Weather; and $T \in \{24, 48, 96, 228, 672\}$ for ETTm1 and ETTm2. The best results are in **bold**, while the second-best are underlined.

Methods		Unsupervised Representation Learning										End-to-end Forecasting			
		TimeDRL		SimTS		TS2Vec		TNC		CoST		Informer		TCN	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	**0.027**	**0.126**	0.036	0.143	0.039	0.151	0.057	0.184	0.040	0.152	0.098	0.147	0.104	0.254
	48	**0.040**	**0.152**	0.054	0.176	0.062	0.189	0.094	0.239	0.060	0.186	0.158	0.319	0.206	0.366
	168	**0.068**	**0.200**	0.084	0.216	0.142	0.291	0.171	0.329	0.097	0.236	0.183	0.346	0.462	0.586
	336	**0.084**	**0.228**	0.100	0.239	0.160	0.316	0.179	0.345	0.112	0.258	0.222	0.387	0.422	0.564
	720	**0.086**	**0.231**	0.126	0.277	0.179	0.345	0.235	0.408	0.148	0.306	0.269	0.435	0.438	0.578
ETTh2	24	**0.070**	**0.205**	0.077	0.206	0.097	0.230	0.097	0.238	0.079	0.207	0.093	0.240	0.109	0.251
	48	**0.097**	**0.241**	0.116	0.259	0.124	0.274	0.131	0.281	0.118	0.259	0.155	0.314	0.147	0.302
	168	**0.166**	**0.323**	0.191	0.340	0.198	0.355	0.197	0.354	0.189	0.339	0.232	0.389	0.209	0.366
	336	**0.177**	**0.340**	0.199	0.354	0.205	0.364	0.207	0.366	0.206	0.360	0.263	0.417	0.237	0.391
	720	0.222	0.378	0.212	0.370	0.208	0.371	0.207	0.370	0.214	0.371	0.277	0.431	**0.200**	**0.367**
ETTm1	24	**0.012**	**0.080**	0.013	0.084	0.016	0.093	0.019	0.103	0.015	0.088	0.030	0.137	0.027	0.127
	48	**0.019**	**0.105**	0.024	0.112	0.028	0.126	0.045	0.162	0.025	0.117	0.069	0.203	0.040	0.154
	96	**0.028**	**0.129**	0.041	0.143	0.045	0.162	0.054	0.178	0.038	0.147	0.194	0.372	0.097	0.246
	288	**0.051**	**0.173**	0.098	0.207	0.095	0.235	0.142	0.290	0.077	0.209	0.401	0.544	0.305	0.455
	672	**0.070**	**0.201**	0.117	0.242	0.142	0.290	0.136	0.290	0.113	0.257	0.277	0.431	0.200	0.367
ETTm2	24	0.024	0.104	**0.022**	**0.099**	0.038	0.139	0.045	0.151	0.027	0.112	0.036	0.141	0.048	0.153
	48	0.048	0.155	**0.045**	**0.149**	0.069	0.194	0.080	0.201	0.054	0.159	0.069	0.200	0.063	0.191
	96	**0.065**	**0.187**	0.068	0.189	0.089	0.225	0.094	0.229	0.072	0.196	0.095	0.240	0.129	0.265
	288	**0.117**	**0.258**	0.160	0.272	0.161	0.306	0.155	0.309	0.153	0.307	0.211	0.367	0.208	0.352
	672	**0.172**	**0.322**	0.249	0.334	0.201	0.351	0.197	0.352	0.183	0.329	0.267	0.417	0.222	0.377
Exchange	24	**0.026**	**0.122**	0.027	0.128	0.033	0.142	0.082	0.227	0.028	0.128	0.103	0.262	0.033	0.139
	48	**0.047**	**0.163**	0.049	0.169	0.059	0.191	0.116	0.268	0.048	0.169	0.121	0.283	0.060	0.188
	168	0.174	**0.309**	**0.158**	0.314	0.180	0.340	0.275	0.411	0.161	0.319	0.168	0.337	0.214	0.350
	336	0.412	**0.485**	**0.382**	0.488	0.465	0.533	0.579	0.582	0.399	0.497	1.672	1.036	0.476	0.527
	720	**1.070**	**0.793**	1.600	1.016	1.357	0.931	1.570	1.024	1.639	1.044	2.478	1.310	1.166	0.830
Weather	24	**0.005**	**0.052**	0.098	0.214	0.096	0.215	0.102	0.221	0.096	0.213	0.117	0.251	0.109	0.217
	48	**0.002**	**0.032**	0.136	0.260	0.140	0.264	0.139	0.264	0.138	0.262	0.178	0.318	0.143	0.269
	168	**0.017**	**0.095**	0.120	0.328	0.207	0.335	0.198	0.328	0.207	0.334	0.266	0.398	0.188	0.319
	336	**0.067**	**0.190**	0.221	0.349	0.231	0.360	0.215	0.347	0.230	0.356	0.197	0.416	0.192	0.320
	720	**0.126**	**0.268**	0.235	0.365	0.233	0.365	0.219	0.353	0.242	0.370	0.359	0.466	0.198	0.329

B.4 Implementation Details

We divide the dataset into three portions: 60% for training, 20% for validation, and 20% for testing, unless a predefined train-test split is provided. For optimization, we use the AdamW optimizer [45] with weight decay. The experiments are carried out on an NVIDIA GeForce RTX 3070 GPU.

The model architecture utilizes a Transformer encoder as the main architecture for f_θ. The timestamp-predictive head p_θ is implemented using a linear layer, while the instance-contrastive head p_θ is designed as a two-layer bottleneck MLP with BatchNorm and ReLU activation in between.

In the time-series forecasting task, we adopt channel-independence along with patching, a concept introduced by PatchTST [26]. This approach treats multivariate time-series as a collection of univariate series, all processed by a single model. Although channel-mixing models leverage cross-channel data directly, channel-independence captures cross-channel interactions indirectly through shared weights. We found that channel-independence greatly improved performance in time-series forecasting, which led to its inclusion in our experiments. However, in the case of time-series classification, omitting channel-independence produced better results.

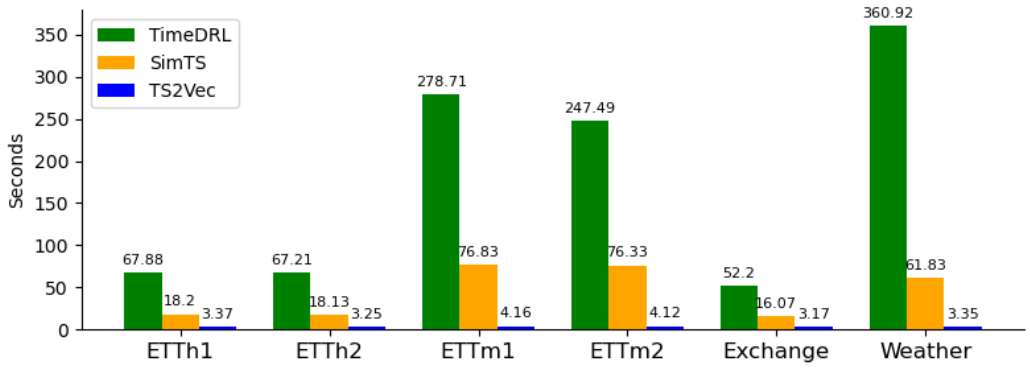

Figure 5: **Comparison of training time (in seconds) in the pre-training stage on forecasting datasets.**

B.5 Linear Evaluation on Time-Series Forecasting

To evaluate the effectiveness of TimeDRL's timestamp-level embeddings, we conduct a linear evaluation on time-series forecasting. This process involves pre-training the encoder using pretext tasks, then freezing the encoder weights and attaching a linear layer for training on the downstream forecasting task. Following the experimental setup from SimTS [11], we define various prediction lengths $T \in \{24, 48, 168, 336, 720\}$ for datasets such as ETTh1, ETTh2, Exchange, and Weather, and $T \in \{24, 48, 96, 228, 672\}$ for ETTm1 and ETTm2. The performance of TimeDRL in multivariate forecasting is summarized in Table 6, where it demonstrates an average MSE improvement of 58.02% over state-of-the-art methods.

Notably, TimeDRL outperforms all baselines, including SimTS, despite SimTS being specifically designed for forecasting tasks with its contrastive learning objective. While SimTS excels at predicting the latent representation of future data from historical data, its performance is constrained by its inability to account for randomness in the data. In contrast, TimeDRL incorporates randomness, enabling it to better capture temporal dynamics and improve prediction accuracy. The lack of randomness in SimTS leads to diminished generalizability, as it fails to capture the inherent variability and unpredictability present in real-world data. This limitation becomes especially evident with longer prediction lengths. For instance, on the ETTh2 dataset with a prediction length of 720, TimeDRL achieves a 78.96% improvement in MSE compared to SimTS, highlighting its substantial enhancement in long-term forecasting accuracy. Additionally, to evaluate the model's capability in univariate time-series forecasting, we conducted experiments on univariate forecasting. As shown in Table 7, TimeDRL shows an average improvement of 29.09% in MSE, further confirming its versatility.

The execution time of TimeDRL, compared to the top-performing baselines SimTS and TS2Vec, is illustrated in Figure 5, with evaluations performed on an NVIDIA GeForce RTX 3070 GPU. For a fair comparison across all methods, we set the batch size to 32, the number of epochs to 10, and the sequence length T to 512. SimTS and TS2Vec utilize fast, convolutional-based encoders, while TimeDRL employs a Transformer encoder, which is known for its superior capability to capture temporal dependencies, though with a longer execution time. To enhance efficiency, TimeDRL integrates a patching mechanism, reducing the input sequence length from L to $\lfloor (L-P)/S \rfloor + 2$, which lowers computational and memory requirements quadratically. Although TimeDRL has a longer execution time compared to its convolutional counterparts, the patching mechanism substantially narrows the efficiency gap, allowing it to capture complex temporal dependencies without sacrificing performance.

B.6 Linear Evaluation on Time-Series Classification

To assess the effectiveness of TimeDRL's instance-level embeddings, we use a linear evaluation method for time-series classification. Similar to the approach in the forecasting evaluation, we first pre-train the encoder using self-supervised learning, freeze the encoder's weights, and then add a

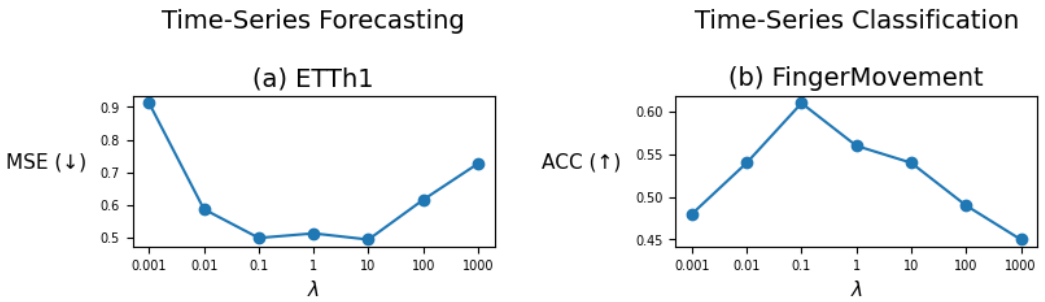

Figure 6: **Sensitivity analysis on λ.** When λ is smaller, the focus is more on predictive loss ($\mathcal{L}_P$); conversely, a larger λ shifts the focus towards contrastive loss ($\mathcal{L}_C$).

linear layer to train on classification tasks. The performance of TimeDRL in time-series classification is shown in Table 3, where it achieves an average accuracy improvement of 1.48For datasets where baseline models already reach around 90Notably, on the more difficult FingerMovements dataset, where baseline models generally struggle, TimeDRL achieves a significant 22.86In addition, TimeDRL demonstrates its capability with univariate data, performing nearly on par with the top baseline method on the Epilepsy dataset, with only a 0.07

B.7 Semi-Supervised Learning

The most practical use of self-supervised learning in real-world applications is in semi-supervised learning, where labeled data are scarce but large amounts of unlabeled data are available. Traditional supervised learning methods focus solely on the limited labeled data, ignoring the potential value of the unlabeled data. Self-supervised learning excels in this scenario by enabling the use of vast amounts of unlabeled data to learn strong representations. In this process, we first train an encoder on a large, unlabeled dataset to extract rich representations and then fine-tune the model using a small set of labeled data alongside the downstream task head. Unlike in linear evaluation, where the encoder weights remain frozen, in this scenario, the encoder weights are fine-tuned during the adjustment phase. To simulate limited labeled data availability, we randomly withhold a portion of the labels from our datasets. The comparison between using only labeled data (supervised learning) and combining both unlabeled and labeled data (TimeDRL with fine-tuning) is depicted in Figure 4. The results show that integrating unlabeled data with TimeDRL significantly enhances performance in both forecasting (measured by MSE) and classification (measured by accuracy), especially when the amount of labeled data is reduced. This trend underscores TimeDRL's ability to utilize unlabeled data effectively, improving performance as labeled data becomes more limited. Interestingly, the advantages of TimeDRL's pre-training phase are evident even when 100

B.8 Ablation Study

B.8.1 Pretext Tasks

In TimeDRL, we purposefully use two pretext tasks to enhance both timestamp-level and instance-level embeddings. The timestamp-predictive task targets loss specifically on timestamp-level embeddings, while the instance-contrastive task focuses on optimizing instance-level embeddings. We conducted a sensitivity analysis of the lambda parameter in $\mathcal{L} = \mathcal{L}_P + \lambda \cdot \mathcal{L}_C$ to evaluate its effect on representation learning. As shown in Figure 6, combining both pretext tasks achieves the best performance in both forecasting and classification tasks, emphasizing the importance of each task in refining the dual-level embeddings. Interestingly, the instance-contrastive task, despite primarily improving instance-level embeddings, significantly enhances performance in time-series forecasting tasks that rely on timestamp-level embeddings. When the instance-contrastive task's contribution is minimal ($\lambda = 0.001$), the MSE increases sharply compared to the scenario where both tasks contribute equally ($\lambda = 1$). A similar trend is observed in the classification task, where prioritizing the instance-contrastive task over the timestamp-predictive task ($\lambda = 1000$) leads to a noticeable drop in accuracy. These results highlight the critical role both pretext tasks play in the overall success of TimeDRL across diverse time-series applications.

Table 8: **Ablation study on data augmentation.** We use the prediction length $T = 168$. The best results are in **bold**.

Data Augmentation	ETTh1	Exchange
None (Ours)	**0.418**	**0.146**
Jitter	0.462 (+10.36%)	0.149 (+2.06%)
Scaling	0.534 (+27.67%)	0.256 (+74.77%)
Rotation	0.703 (+68.15%)	0.402 (+174.46%)
Permutation	0.607 (+45.09%)	0.199 (+35.73%)
Masking	0.438 (+4.76%)	0.160 (+9.47%)
Cropping	0.462 (+10.57%)	0.216 (+47.70%)

Table 9: **Ablation study on pooling methods.** The best results are in **bold**.

Pooling Method	FingerMovements	Epilepsy
[CLS] (Ours)	**63.00**	**95.83**
Last	57.00 (-9.52%)	79.78 (-16.75%)
GAP	51.00 (-19.05%)	79.78 (-16.75%)
All	60.00 (-4.76%)	79.78 (-16.75%)

B.8.2 Data Augmentations

The core design principle of TimeDRL is to avoid any data augmentation in order to prevent the introduction of inductive biases. As a result, neither the timestamp-predictive task nor the instance-contrastive task relies on augmentation methods. In this experiment, we aimed to showcase the potential negative impact of overlooking inductive bias. In Table 8, we experiment with six time-series-specific data augmentation techniques [46, 10]. **Jittering** introduces sensor noise through additive Gaussian noise. **Scaling** modifies data magnitude by multiplying it with a random scalar. **Rotation** permutes the order of features and can flip the sign of feature values. **Permutation** divides the data into segments and rearranges them randomly to create new time-series instances. **Masking** randomly sets portions of the time-series data to zero. **Cropping** removes sections from the left and right of a time-series instance and fills the gaps with zeros to maintain the original sequence length.

In Table 8, the use of any augmentation method led to reduced performance, with an average MSE increase of 27.77% for the ETTh1 dataset and 57.37% for the Exchange dataset. The largest performance drop was observed with the Rotation augmentation, which resulted in a 68.15% increase in MSE for the ETTh1 dataset and 174.46% for the Exchange dataset. Although TS2Vec [10] addresses the issue of inductive bias, it still employs Masking and Cropping augmentations. Our results indicate that while these two methods are less detrimental than others, they still degrade performance. This experiment supports our hypothesis that completely avoiding augmentation methods is crucial for eliminating inductive bias and ensuring TimeDRL's optimal performance.

B.8.3 Pooling Methods

In TimeDRL, we use a dedicated [CLS] token strategy to obtain instance-level embeddings directly from the patched time-series data. However, we acknowledge the possibility of extracting instance-level embeddings from timestamp-level embeddings using various pooling methods. To explore this, we conducted experiments with three alternative pooling strategies for instance-level embeddings, as shown in Table 8. **Last** takes the last timestamp-level embedding as the instance-level representation. **GAP** (Global Average Pooling) averages the timestamp-level embeddings across the time axis to generate the instance-level embedding. **All** flattens all timestamp-level embeddings to form a single instance-level representation.

The results in Table 9 show that using pooling methods other than TimeDRL's [CLS] token strategy leads to an average accuracy drop of 11.11% on the FingerMovements dataset and 16.75The least effective pooling method is GAP, a commonly used method in the time-series domain [10], which suffers the most significant performance decline due to the anisotropy problem. These findings

Table 10: **Ablation study on the architecture of the backbone encoder.** We use the prediction length $T = 168$. The best results are in **bold**.

Backbones	ETTh1	Exchange
Transformer Encoder (Ours)	**0.418**	**0.146**
Transformer Decoder	0.465 (+11.26%)	0.159 (+8.28%)
ResNet	0.576 (+37.76%)	0.160 (+9.50%)
TCN	0.517 (+23.72%)	0.148 (+1.13%)
LSTM	0.451 (+7.84%)	0.160 (+9.46%)
Bi-LSTM	0.443 (+5.92%)	0.153 (+4.62%)

Table 11: **Ablation study on the stop gradient operation.** The best results are in **bold**.

Stop Gradient	FingerMovements	Epilepsy
TimeDRL	**63.00**	**95.83**
TimeDual w/o SG	56.00 (-11.11%)	79.78 (-16.75%)

underscore the importance of separating timestamp-level and instance-level embeddings, a crucial element in TimeDRL's enhanced performance compared to other baseline methods.

B.8.4 Encoder Architectures

Transformers are widely recognized for their success in downstream time-series tasks [47, 26], but in self-supervised learning for time-series, CNN-based [21, 10] and RNN-based [12] models are more frequently used than Transformers. TimeDRL is specifically designed to leverage the strengths of Transformers in self-supervised learning for time-series data, aiming to showcase the Transformer's capabilities in this domain. In TimeDRL, the Transformer encoder serves as the core architecture. To benchmark its performance against other encoders, we conducted experiments using five different models, as shown in Table 10. **Transformer Decoder** adopts a structure similar to the Transformer encoder but uses masked self-attention, ensuring that each timestamp can only attend to prior timestamps, preventing access to future information. **ResNet** adapts the ResNet18 architecture from computer vision by employing one-dimensional convolutions suited for time-series data. **TCN** [43] combines dilations and residual connections with causal convolutions, making it well-suited for autoregressive prediction in time-series tasks. **LSTM** uses Long Short-Term Memory units to capture sequential dependencies. In this case, a uni-directional LSTM is used, focusing on past and present data to avoid future data leakage. **Bi-LSTM** follows the LSTM architecture but incorporates bi-directional processing, enabling the model to access both past and future timestamps.

The results in Table 10 indicate that not using the Transformer encoder in combination with the two pretext tasks leads to decreased performance. This results in an average MSE increase of 17.30% for the ETTh1 dataset and 6.60% for the Exchange dataset, underscoring the superior performance of the Transformer encoder. In comparison, the Transformer Decoder shows a performance drop, with an 11.26% increase in MSE for the ETTh1 dataset and 8.28% for the Exchange dataset, emphasizing the importance of bidirectional self-attention for a thorough understanding of the entire sequence. Similarly, when comparing LSTM with Bi-LSTM, the latter outperforms the former due to its ability to process both past and future information. These results highlight the significance of full temporal access at each timestamp, confirming the Transformer encoder's effectiveness in capturing robust time-series representations.

B.8.5 Stop Gradient

To address sampling bias, TimeDRL incorporates an additional prediction head with a stop-gradient operation. This asymmetric design, with one path using the extra prediction head and the other utilizing the stop-gradient, is effective in preventing model collapse, as supported by prior work [9, 48]. The results in Table 11 demonstrate that removing the stop-gradient operation leads to a notable accuracy drop of 11.11% on the FingerMovements dataset and 16.75% on the Epilepsy dataset, emphasizing the critical role the stop-gradient mechanism plays in this asymmetric architecture.