GIVING SENSORS A VOICE: MULTIMODAL JEPA FOR SEMANTIC TIME-SERIES EMBEDDINGS

Utsav Dutta*
C3 AI
1400 Seaport Blvd
Redwood City, CA 94063
utsav.dutta@c3.ai

Gerardo Pastrana*
C3 AI
1400 Seaport Blvd
Redwood City, CA 94063
gerardo.pastrana@c3.ai

Sina Khoshfetrat Pakazad
C3 AI
1400 Seaport Blvd
Redwood City, CA 94063
sina.pakazad@c3.ai

Henrik Ohlsson
C3 AI
1400 Seaport Blvd
Redwood City, CA 94063
henrik.ohlsson@c3.ai

ABSTRACT

We introduce CHARM (Channel-Aware Representation Model), a multimodal architecture for self-supervised time series representation learning that incorporates channel-level textual descriptions into both temporal convolutional and attention layers. This enables the model to reason about sensor identity and inter-channel relationships while remaining invariant to channel ordering. Trained with a Joint Embedding Predictive Architecture (JEPA), CHARM learns temporally stable, noise-robust embeddings by predicting in latent space rather than reconstructing raw signals. Across classification, forecasting, and anomaly detection benchmarks, CHARM's frozen embeddings with a lightweight linear probe match or outperform significantly larger task-specific foundation models.

Track: Research

1 INTRODUCTION

Time series models are critical across manufacturing, energy, healthcare, and finance, yet most remain task-specific and require significant manual effort (Hannun et al., 2019; Susto et al., 2014; Ding et al., 2015). While foundation models have transformed NLP and vision through self-supervised pretraining on large unlabeled corpora (Devlin et al., 2019; Assran et al., 2023; Brown et al., 2020), analogous efforts in time series have focused primarily on supervised forecasting (Das et al., 2024; Woo et al., 2024b; Ansari et al., 2024), yielding representations that are specialized and brittle across tasks. Self-supervised alternatives such as MOMENT (Goswami et al., 2024) and UniTS (Gao et al., 2024a) rely on masked reconstruction, which entangles learned representations with sensor-level noise (Trirat et al., 2024). Joint Embedding Predictive Architectures (JEPA) (LeCun, 2022) offer a more robust paradigm by predicting in latent space, filtering noise and capturing higher-level temporal structure (Assran et al., 2023). Orthogonally, nearly all time series models treat input channels as anonymous streams, discarding valuable sensor metadata that domain experts routinely use for interpretation.

To address these gaps, we introduce **CHARM** (**CH**annel-**A**ware **R**epresentation **M**odel), a foundation embedding model that jointly leverages raw signals and textual channel descriptions. Our core contributions are:

1. **Description-aware temporal featurization.** We modify TCN layers to generate convolutional kernels and gates from channel descriptions, eliminating manual patch-size tuning (§2.1.1).

*Equal contribution

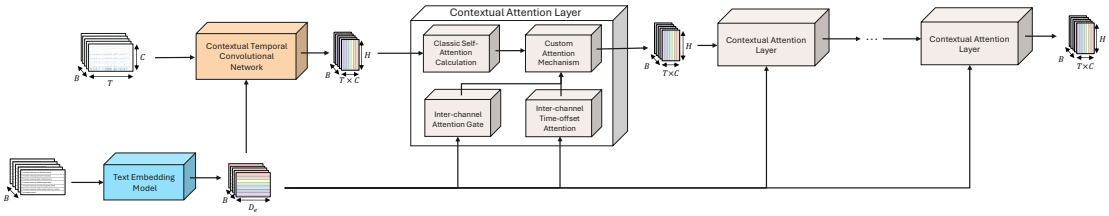


Figure 1: Overview of the CHARM architecture: a contextual TCN followed by contextual attention layers, each guided by textual channel descriptions. Notation is provided in Section B.

2. **Description-conditioned attention and gating.** Novel inter-channel attention layers and gating modules, conditioned on text, capture cross-channel dependencies while maintaining channel-order invariance (§2.1.2).
3. **JEPA-based self-supervised training for time series.** We adapt JEPA with tailored augmentations and a multi-granularity loss that promotes temporally stable embeddings without reconstruction (§2.2).

We evaluate CHARM on classification, forecasting, and anomaly detection, where it consistently achieves strong performance across diverse datasets.

2 Methodology

Figure 1 presents an overview of CHARM's architecture. We first describe the model (§2.1) and then the self-supervised training procedure (§2.2). Full notation is in Section B.

2.1 Multi-Modal Time Series Embedding Model

CHARM combines a description-aware temporal convolutional network (§2.1.1) with a stack of contextual attention layers (§2.1.2), as shown in Figure 1.

2.1.1 Contextual Temporal Convolutional Network

We introduce a contextual TCN that projects $\mathbf{T} \in \mathbb{R}^{T \times C}$ into embeddings $\mathbf{T}_c \in \mathbb{R}^{T \times C \times H}$. Standard dilated TCNs (Bai et al., 2018; Lin et al., 2021) use fixed, input-independent kernels, limiting adaptability across heterogeneous domains. We make the TCN *context-aware* by conditioning on channel description embeddings $\mathbf{E}_d \in \mathbb{R}^{C \times D_e}$ (extracted via a frozen text encoder) through two mechanisms:

Contextual kernel gating. Soft gates $\mathbf{G}_c = \text{sigmoid}(\mathbf{E}_d \mathbf{W}_g)$, $\mathbf{W}_g \in \mathbb{R}^{D_e \times N}$, control the effective receptive field of each channel across N TCN layers (see Figure 3).

Contextual kernels. Rather than fixed filters, convolutional kernels are generated from descriptions: $\mathbf{G}_k = \mathbf{E}_d \mathbf{W}_k$, $\mathbf{W}_k \in \mathbb{R}^{D_e \times (HK/N)}$, directly tying channel semantics to filter generation (see Figure 3).

2.1.2 Contextual Attention Layer

The TCN embeddings are processed by contextual attention layers that fuse channel and temporal dimensions via novel extensions to self-attention (Vaswani et al., 2017), conditioned on channel descriptions.

Description-aware inter-channel gating. Given $\mathbf{E}_d$, we compute pairwise similarities $\mathbf{S} = \mathbf{E}_d \mathbf{E}_d^\top$ and a learnable threshold matrix $\mathbf{Z}[i,j] = \text{sigmoid}(\mathbf{E}_d[i,:]\, \mathbf{W}_b\, \mathbf{E}_d[j,:]^\top)$. The gating matrix $\mathbf{G}_d = \text{ReLU}(\mathbf{Z} - \mathbf{S})$ selectively suppresses cross-channel attention (see Figure 4).

Description-aware inter-channel time-offset attention. A learnable tensor $\mathbf{\Delta} \in \mathbb{R}^{C \times C \times 2T_{\text{max}}}$ encodes dependency strength between channel i and channel j at temporal offset t. It is constructed from pairwise description embeddings projected via $\mathbf{W}_d \in \mathbb{R}^{2D_e \times T_{\text{max}}}$ and symmetrized so that

$\mathbf{\Delta}_{i,j,t} = \mathbf{\Delta}_{j,i,-t}$ (see Figure 5). The final $\bar{\mathbf{\Delta}} \in \mathbb{R}^{CT \times CT}$ is assembled via a "slice-and-tile" operation (Section C.6.1).

Custom attention mechanism. We unify the above into a single attention framework. The TCN embeddings are reshaped to $\mathbf{X} \in \mathbb{R}^{CT \times H}$ and RoPE is applied to queries and keys. The attention matrix combines vanilla self-attention, channel-lag offsets, and channel gates:

$$\mathbf{A}_{[(i,p),(j,q)]} = \text{Softmax}\left(\underbrace{\frac{\hat{\mathbf{Q}}_{[i,p,:]}\hat{\mathbf{K}}^T_{[j,q,:]}}{\sqrt{D_e}}}_{\text{Self-Attention}} + \underbrace{\mathbf{\Delta}[i,j,q-p]}_{\text{Channel Lags}} - \underbrace{\lambda_G \mathbf{G}_d[i,j]}_{\text{Channel Gates}} \right)$$

where λ_G is a large scalar enabling the gate to act as a soft attention mask.

The complete encoder, denoted $\mathbf{E}_\theta$, stacks the contextual TCN with N contextual attention layers and applies ℓ_2 normalization to the final embeddings (see Section C for details).

2.2 Self-supervised Representation Learning

We adopt the JEPA framework (Assran et al., 2023; LeCun, 2022) to train CHARM via self-supervised learning. JEPA comprises context, target, and predictor encoders. The context and target encoders share the CHARM architecture; the target is updated via exponential moving average. The predictor is a narrower variant that generates embeddings for masked positions from context embeddings. We generate augmented views of each time series instance via two self-supervised tasks (*causal prediction* and *smoothing*) combined with perturbations for robustness. Training minimizes a multi-resolution ℓ_1 loss that aligns predictor and target embeddings at three granularities (per-channel-per-timestep, per-timestep, and global), supplemented by regularization terms that encourage sparsity in the learned channel gating and time-offset tensors. The full loss formulation is provided in Section D; JEPA setup, encoder integration, and dataset generation details are in Section E.

3 Experiments

In this section, we evaluate our model's embeddings on common downstream tasks, namely classification, forecasting, and anomaly detection, and benchmark our model against the current state-of-the-art models in each of the aforementioned downstream tasks. We expand on our datasets in Section I, and provide more details on downstream task training and baselines in Section K and Table 9 respectively.

Forecasting We evaluate forecasting on the LSF benchmark suite (Wu et al. 2021, ETTh1/2, ETTm1/2, Weather) in the standard multi-horizon, multivariate setting with horizons 96/192/336/720. We compare against state-of-the-art foundation time-series models that are either (i) pre-trained for forecasting or (ii) pre-trained for representation learning and evaluated via dataset-specific linear probing (LP).

To produce point forecasts with CHARM, we consider three variants: (1) **CHARM+LP** — dataset-specific linear probes trained on frozen CHARM embeddings; (2) **CHARM + NLH** — a single, dataset-agnostic non-linear forecaster trained on frozen CHARM embeddings; and (3) **CHARM + NLH FT** — end-to-end training that back-propagates through CHARM to align the embeddings with the point-forecast objective. Per-dataset means (averaged across horizons) are reported in Table 1.

Dataset	Toto		Moirai_Base		TimeMixer++		VisionTS		MOMENT-LP		PatchTST-LP		CHARM					
													CHARM + LP		CHARM+NLH		CHARM+NLH FT	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	<u>0.224</u>	**0.245**	0.238	0.261	0.226	0.262	0.269	0.292	0.228	0.266	0.233	0.270	0.230	0.262	0.230	0.262	**0.222**	<u>0.255</u>
ETTm1	0.396	<u>0.378</u>	0.382	0.404	0.368	<u>0.378</u>	0.373	**0.371**	**0.344**	0.379	<u>0.350</u>	0.382	0.413	0.432	0.416	0.437	0.411	0.428
ETTm2	0.266	0.303	0.302	**0.295**	0.269	0.320	0.281	0.321	0.259	0.318	0.262	0.322	<u>0.209</u>	<u>0.298</u>	0.220	0.304	**0.208**	0.299
ETTh1	0.435	<u>0.413</u>	0.432	0.440	<u>0.395</u>	0.419	**0.392**	**0.405**	0.418	0.422	0.428	0.438	0.554	0.532	0.592	0.555	0.557	0.526
ETTh2	0.349	**0.363**	0.346	0.382	0.339	0.380	0.333	<u>0.374</u>	0.352	0.395	0.365	0.386	0.324	0.386	<u>0.323</u>	0.385	**0.316**	0.381

Table 1: Per-dataset mean MSE/MAE (lower is better). **Bold** = best, <u>underline</u> = second-best. The last three methods are grouped as **Ours**: CHARM-LP, *CHARM + Non-linear Head*, and *Fully Fine-tuned*. Notably, **CHARM-LP** frequently outperforms much larger foundation models, highlighting the parameter efficiency of channel-aware representations.

Remarkably, even under the frozen-embedding linear probe setup, CHARM outperforms significantly larger, billion-parameter models like Moirai and Toto on ETTm2 and ETTh2. With **minimal** end-to-end fine-tuning, CHARM attains the lowest MSE on 3/5 datasets, establishing it as a highly efficient alternative to massive forecasting-specific foundation models. By contrast, MAE leaders skew toward models optimized with MAE-aligned objectives (e.g., *Toto*, *VisionTS*, and *Morai*), which explains our relative MAE gap under an MSE-optimized head. See Section K.3 for forecasting-head details, training setup. Disaggregated results can be found in Table 22.

Classification We evaluate our model on multivariate time series classification using the UEA dataset (Bagnall et al., 2018), benchmarking against semantic and reconstruction-based representation learning methods as well as specialized classification models. Results are summarized in Table 2.

We test two protocols: (i) frozen encoder embeddings passed to an SVM with an `rbf` kernel, and (ii) a finetuned encoder with a linear classification head trained via cross-entropy loss. Finetuning yields substantial gains over the frozen setting and baselines, achieving the highest number of wins, average accuracy, and unnormalized correct predictions. Against MiniRocket—a SOTA task-specific method—our frozen model is competitive despite MiniRocket's stronger raw scores (see Table 12), showing that our pretraining produces strong embeddings without task-specific adaptation. With finetuning, performance improves across all metrics, particularly in unnormalized scores (see Table 14), providing evidence that post-training alignment can substantially enhance downstream task performance.

Method	Wins↑	Avg. Acc.↑	Total Correct↑
TS2Vec	2	78.1	7467
T-Loss	3	72.8	7141
TS-TCC	2	74.4	7118
T-Rep	3	78.5	7363
MOMENT	3	72.5	5414
MiniROCKET	4	77.6	7569
$\text{CHARM}_{\text{frozen+SVM}}$	4	79.6	7431
$\text{CHARM}_{\text{finetuned}}$	**5**	**80.9**	**7799**

Table 2: **Multivariate Classification Results. CHARM** achieves the highest average accuracy and total wins; notably, our frozen embeddings alone outperform specialized representation learning baselines, demonstrating the discriminative power of channel-aware features.

Anomaly Detection We use two tasks to evaluate our model's performance on anomaly detection. We use the Skoltech Anomaly Detection Benchmark (SKAB), to assess performance on real world multivariate datasets, as well as the popular UCR univariate anomaly detection dataset. For i) SKAB, we use baselines that consist of classical anomaly detection, CNN/LSTM based models, as well as more recent representation learning models. We reproduce these baselines by training a linear reconstruction head on each of the 34 datasets in SKAB, and evaluating on the corresponding test instances. The evaluation setup in the SKAB test suite is uniformly applied to all models, which relies on using the errors at each time point in the test set to classify anomalies based on selecting an appropriate threshold (Section K.2). The setup for the reconstruction head (including optimization setup) is identical for all baselines for which the SKAB results were obtained by training[1]. As seen in Table 3, CHARM has the highest F1 score, followed closely by MOMENT, demonstrating strong performance on multivariate anomaly detection.

Method	F1↑	FAR↓	MAR↓
MSCRED	0.36	49.94	69.88
T-Rep	0.78	**12.60**	28.51
MOMENT_0	0.79	14.20	26.98
TS2Vec	0.79	12.77	27.61
$\text{MOMENT}_{\text{LP}}$	<u>0.82</u>	15.52	20.73
CHARM	**0.86**	19.35	**12.69**

Table 3: **Multivariate Anomaly Detection (34 SKAB Datasets). CHARM** achieves the highest F1 score and the lowest Missed Alarm Rate (MAR) among all baselines. These results highlight the benefits of JEPA-driven latent representations for distinguishing critical process anomalies from sensor noise.

For (ii) UCR, we train our model with a reconstruction head on 46 UCR univariate anomaly detection datasets, which come from a diverse set of domains with varying types of anomalies. We benchmark our model against both task-specific anomaly detection methods and representation learning approaches, using the average adjusted F1 score as the standard evaluation metric. We report the per dataset scores, and wins in Table 15. Table 4 shows that CHARM has the highest average F1 score across all datasets.

Method	F1↑
Anomaly Transformer	0.485
DGHL	0.415
GPT4TS	0.479
TimesNet	0.627
MOMENT	0.684
CHARM	**12.69**

Table 4: **Univariate Anomaly Detection (46 UCR Datasets). CHARM** outperforms task-specific architectures and large-scale foundation models, highlighting strong zero-shot transfer.

[1] MOMENT_0 is used directly in reconstruction mode with no training, whereas $\text{MOMENT}_{\text{LP}}$'s reconstruction head is trained for each SKAB dataset, similar to other models.

Ablations We ablate (i) the TCN featurization layer and (ii) the text-conditioned attention mechanism (Table 24 in Section L). Replacing standard patch embeddings with a text-free TCN already improves classification accuracy from 63.0% to 68.0%; adding text-conditioned convolutions further raises it to 71.4% while cutting forecasting MSE nearly in half. In the attention layer, combining channel gating ($\mathbf{G}$) with time-offset attention ($\mathbf{\Delta}$) yields the best results across all metrics, confirming that both components are complementary. The learned channel gates also reveal interpretable structure: on the ETT dataset, a causal hierarchy emerges where the target variable (`Oil Temperature`) attends to all independent channels but not vice versa (see Section M.2).

To disentangle the contributions of different factors, we conduct additional controlled ablations (Table 29 in Section L.1). The JEPA objective contributes the largest single-factor effect (-4.4 pp when replaced with reconstruction). The conditioning mechanism contributes $+5.0$ pp over naive additive fusion, even when text is replaced with fixed channel-index vectors. Shuffled descriptions (-2.6 pp) degrade more than random embeddings (-0.7 pp), confirming the model exploits correct channel–text correspondence. Text semantics provide a measurable forecasting advantage (0.4117 vs. 0.4138 avg MSE) and enable cross-dataset transfer to unseen sensor configurations.

REFERENCES

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *Transactions on Machine Learning Research*, 2024.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision transformer, 2021. URL `https://arxiv.org/abs/2103.15691`.

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture, 2023. URL `https://arxiv.org/abs/2301.08243`.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

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, 2018. URL `https://arxiv.org/abs/1811.00075`.

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.

Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning, 2023. URL `https://arxiv.org/abs/2304.12210`.

Adrien Bardes, Jean Ponce, and Yann LeCun. Mc-jepa: A joint-embedding predictive architecture for self-supervised learning of motion and content features. *arXiv preprint arXiv:2307.12698*, 2023. URL `https://doi.org/10.48550/arXiv.2307.12698`.

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video, 2024. URL `https://arxiv.org/abs/2404.08471`.

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.

Yifu Cai, Mononito Goswami, Arjun Choudhry, Arvind Srinivasan, and Artur Dubrawski. Jolt: Jointly learned representations of language and time-series. In *Workshop on Deep Generative Models for*

Health, NeurIPS 2023, 2023. URL https://openreview.net/pdf?id=UVF1AMBj9u. poster.

Mouxiang Chen, Lefei Shen, Zhuo Li, Xiaoyun Joy Wang, Jianling Sun, and Chenghao Liu. Visionts: Visual masked autoencoders are free-lunch zero-shot time series forecasters, 2025. URL https://arxiv.org/abs/2408.17253.

Ben Cohen, Emaad Khwaja, Youssef Doubli, Salahidine Lemaachi, Chris Lettieri, Charles Masson, Hugo Miccinilli, Elise Ramé, Qiqi Ren, Afshin Rostamizadeh, Jean Ogier du Terrail, Anna-Monica Toon, Kan Wang, Stephan Xie, Zongzhe Xu, Viktoriya Zhukova, David Asker, Ameet Talwalkar, and Othmane Abou-Amal. This time is different: An observability perspective on time series foundation models, 2025. URL https://arxiv.org/abs/2505.14766.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 10148–10167, 2024.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019. doi: 10.1109/JAS.2019.1911747.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019.

Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *IJCAI*, volume 15, pp. 2327–2333, 2015.

Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5884–5888, 2018. doi: 10.1109/ICASSP.2018.8462506.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Katrina Drozdov, Ravid Shwartz-Ziv, and Yann LeCun. Video representation learning with joint-embedding predictive architectures. *arXiv preprint arXiv:2412.10925*, 2024. URL https://doi.org/10.48550/arXiv.2412.10925.

Thomas D. P. Edwards, James Alvey, Justin Alsing, Nam H. Nguyen, and Benjamin D. Wandelt. Scaling-laws for large time-series models, 2025. URL https://arxiv.org/abs/2405.13867.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

Philip J. Fleming and John J. Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Commun. ACM*, 29:218–221, 1986. URL https://api.semanticscholar.org/CorpusID:1047380.

Archibald Felix Fraikin, Adrien Bennetot, and Stephanie Allassonniere. T-Rep: Representation learning for time series using time-embeddings. In *The 12th International Conference on Learning Representations*, 2024.

Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. In *Advances in neural information processing systems*, volume 32, pp. 4650–4661, 2019.

Shanghua Gao, Teddy Koker, Owen Queen, Thomas Hartvigsen, Theodoros Tsiligkaridis, and Marinka Zitnik. UniTS: a unified multi-task time series model, 2024a. URL https://arxiv.org/abs/2403.00131.

Shanghua Gao, Teddy Koker, Owen Queen, Tom Hartvigsen, Theodoros Tsiligkaridis, and Marinka Zitnik. UniTS: A unified multi-task time series model. In *Advances in Neural Information Processing Systems*, volume 37, pp. 140589–140631, 2024b.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Yuan Gong, Yu-An Chung, and James Glass. AST: audio spectrogram transformer, 2021. URL https://arxiv.org/abs/2104.01778.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. MOMENT: a family of open time-series foundation models, 2024. URL https://arxiv.org/abs/2402.03885.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020. URL https://doi.org/10.48550/arXiv.2006.07733.

Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine*, 25(1):65–69, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

Romain Ilbert, Ambroise Odonnat, Vasilii Feofanov, Aladin Virmaux, Giuseppe Paolo, Themis Palpanas, and Ievgen Redko. SAMformer: Unlocking the potential of transformers in time series forecasting with sharpness-aware minimization and channel-wise attention, 2024. URL https://arxiv.org/abs/2402.10198.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-llm: Time series forecasting by reprogramming large language models, 2024. URL https://arxiv.org/abs/2310.01728.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Iurii D. Katser and Vyacheslav O. Kozitsin. Skoltech anomaly benchmark (SKAB). https://www.kaggle.com/dsv/1693952, 2020.

Jongseon Kim, Hyungjoon Kim, HyunGi Kim, Dongjun Lee, and Sungroh Yoon. A comprehensive survey of deep learning for time series forecasting: Architectural diversity and open challenges. *Artificial Intelligence Review*, 58, 2025.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4015–4026, 2023.

Yann LeCun. A path towards autonomous machine intelligence. *Open Review*, 62(1):1–62, 2022.

Yang Lin, Irena Koprinska, and Mashud Rana. Temporal convolutional attention neural networks for time series forecasting. In *2021 International joint conference on neural networks (IJCNN)*, pp. 1–8, 2021.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. iTransformer: Inverted transformers are effective for time series forecasting. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024.

Yiqin Nie, Zhaohan Wang, Zhiwei Li, Yujing Zhang, Yao Zhang, Yuxuan Wang, Yafeng Wang, and Philip S. Yu. A time series is worth 64 words: Long-term forecasting with transformers. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023a.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023b. URL https://arxiv.org/abs/2211.14730.

Zach Nussbaum, John Xavier Morris, Andriy Mulyar, and Brandon Duderstadt. Nomic embed: Training a reproducible long context text embedder. *Transactions on Machine Learning Research*, 2025.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious, 2025. URL https://arxiv.org/abs/2501.00656.

Zijie Pan, Yushan Jiang, Sahil Garg, Anderson Schneider, Yuriy Nevmyvaka, and Dongjin Song. $\mathbf{S}^2$ip-llm: Semantic space informed prompt learning with llm for time series forecasting, 2024. URL https://arxiv.org/abs/2403.05798.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, 2018. OpenAI.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763, 2021.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36: 1181–1191, 2020.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding, 2020. URL https://arxiv.org/abs/2004.09297.

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, Hongyan Li, Yaliang Li, and Shenda Hong. Test: Text prototype aligned embedding to activate llm's ability for time series, 2024. URL https://arxiv.org/abs/2308.08241.

Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE transactions on industrial informatics*, 11(3):812–820, 2014.

Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. In *International Conference on Learning Representations (ICLR)*, 2021.

Patara Trirat, Yooju Shin, Junhyeok Kang, Youngeun Nam, Jihye Na, Minyoung Bae, Joeun Kim, Byunghyun Kim, and Jae-Gil Lee. Universal time-series representation learning: A survey, 2024. URL https://arxiv.org/abs/2401.03717.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017.

Khang H. N. Vo, Duc P. T. Nguyen, Thong Nguyen, and Tho T. Quan. Ti-jepa: An innovative energy-based joint embedding strategy for text-image multimodal systems. *arXiv preprint arXiv:2503.06380*, 2025. URL https://doi.org/10.48550/arXiv.2503.06380.

Shiyu Wang, Jiawei Li, Xiaoming Shi, Zhou Ye, Baichuan Mo, Wenze Lin, Shengtong Ju, Zhixuan Chu, and Ming Jin. Timemixer++: A general time series pattern machine for universal predictive analysis, 2025. URL https://arxiv.org/abs/2410.16032.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020. URL https://arxiv.org/abs/2002.10957.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers, 2024a. URL https://arxiv.org/abs/2402.02592.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 53140–53164, 2024b.

Haixu Wu, Yao Xu, Jindong Wang, Guodong Long, Chengqi Wang, and Lijun Yao. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pp. 22419–22430, 2021.

Ziheng Yue, Yuxuan Zhang, Yifan Sun, Yifan Wang, and Zenglin Huang. TS2Vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8075–8083, 2022.

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *International Conference on Learning Representations*, 2023.

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 35th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 11106–11115, 2021a.

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, 2021b. URL https://arxiv.org/abs/2012.07436.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting, 2022. URL https://arxiv.org/abs/2201.12740.

Yunjiao Zhou, Jianfei Yang, Han Zou, and Lihua Xie. Tent: Connect language models with iot sensors for zero-shot activity recognition, 2023. URL https://arxiv.org/abs/2311.08245.

A Related Work

Historically, recurrent neural network architectures such as RNNs, LSTMs, and GRUs dominated time series modeling by capturing temporal dependencies through recursive hidden-state updates, achieving success across diverse tasks (Salinas et al., 2020). However, their sequential nature impeded parallelization, leading to slow training and difficulties in modeling long-range dependencies (Kalchbrenner et al., 2016; Pascanu et al., 2013; Zhou et al., 2021a; Kim et al., 2025).

With the emergence of Transformer architectures (Vaswani et al., 2017), significant advancements have occurred across multiple modalities, including images (Dosovitskiy et al., 2021), audio (Gong et al., 2021), video (Arnab et al., 2021), text (Devlin et al., 2018; Radford et al., 2018), and speech (Dong et al., 2018). Inspired by these successes, the time series community has adopted Transformer-based approaches, leading to notable innovations tailored specifically for temporal data (Zhou et al., 2021b; 2022; Ilbert et al., 2024; Wu et al., 2021; Zhang & Yan, 2023).

Simultaneously, self-supervised representation learning (SSRL), widely successful in domains such as vision and language, has demonstrated potential for extracting high-quality embeddings from vast amounts of unlabeled data. These embeddings facilitate downstream tasks—such as forecasting, classification, and anomaly detection—through lightweight task-specific heads. Analogous approaches have been adapted for time series, predominantly using contrastive self-supervised tasks (Yue et al., 2022; Fraikin et al., 2024; Franceschi et al., 2019; Tonekaboni et al., 2021). However, existing approaches typically produce models tailored to specific datasets, limiting their generalizability across arbitrary data sizes or channel configurations.

More recently, foundational models have revolutionized representation learning across natural language processing, computer vision, and audio (Devlin et al., 2019; Nussbaum et al., 2025; Assran et al., 2023; Kirillov et al., 2023; Baevski et al., 2020; Brown et al., 2020; Radford et al., 2021). In time series analysis, considerable progress has focused primarily on forecasting tasks (Das et al., 2024; Woo et al., 2024b; Ansari et al., 2024; Liu et al., 2024). Early foundational attempts predominantly addressed univariate series (Das et al., 2024; Ansari et al., 2024), though recent advancements have successfully extended to multivariate settings with sophisticated cross-channel modeling techniques (Woo et al., 2024b; Liu et al., 2024). Some more recent papers Cohen et al. (2025) have gone to great lengths to fully leverage the scaling laws observed in foundational time series models Edwards et al. (2025) in order to maximize their performance. Foundation embedding models specifically targeting time series representation learning have begun to emerge, leveraging reconstruction-based or next-step forecasting objectives (Goswami et al., 2024; Gao et al., 2024a; Trirat et al., 2024). However, these approaches either focus on univariate series or treat multivariate data as independent channels, inadequately capturing complex inter-channel dynamics. This substantially limits the representational richness and effectiveness of these models in realistic scenarios. See Table 5 for an overview of capabilities of key time series models.

Joint-Embedding Predictive Architectures have found notable success in visual domains by shifting the learning objective from pixel-level reconstruction to latent-space prediction. Extending this approach to video, Meta AI's Video JEPA with Variance–Covariance Regularization (VJ-VCR) (Drozdov et al., 2024) predicts future frame embeddings in a learned representation space while enforcing variance and covariance constraints to prevent collapse; this model outperforms generative reconstruction baselines on downstream tasks such as action recognition and video retrieval by capturing high-level spatiotemporal dynamics. Extensions such as MC-JEPA (Bardes et al., 2023) further demonstrate JEPA's flexibility by jointly learning motion (optical flow) and content features within a shared encoder–predictor framework, matching or surpassing unsupervised optical flow benchmarks and improving downstream segmentation tasks. In multimodal settings, TI-JEPA (Vo et al., 2025) integrates an energy-based JEPA with cross-modal encoders to align text and image embeddings, achieving superior results on multimodal sentiment analysis and visual question answering benchmarks by capturing complex semantic correspondences without reconstructing raw inputs. Complementing JEPA, bootstrapped embedding SSL methods like BYOL ("Bootstrap Your Own Latent") (Grill et al., 2020) train an online network to predict the target network's representation of differently augmented views—updating the target via momentum averaging—and achieve strong results on ImageNet under linear evaluation without requiring negative pairs; this demonstrates that simple latent-space prediction objectives can match or exceed contrastive and reconstruction-based approaches in learning robust, generalizable representations. Together, these concrete instantiations

highlight JEPA's core advantage of filtering out low-level noise and focusing learning on high-level semantic structure, while bootstrapped SSL offers a practical, decoder-free paradigm for self-supervised representation learning, and motivate further exploration of these methods for time series.

Multimodal text + time series models Recent works have explored combining textual information with time series data through several novel approaches. For instance, Jin et al. (2024), Pan et al. (2024), and Sun et al. (2024) reprogram pretrained LLMs to handle time series input. These approaches fall under the *LLM-for-TS* or *TS-for-LLM* paradigms, where either an LLM is finetuned for time series data or the time series are transformed into token sequences consumable by an LLM. However, such methods do not directly leverage textual metadata; rather, they exploit the language modeling capabilities of models pretrained on large corpora of text. In contrast, Zhou et al. (2023) and Cai et al. (2023) explicitly incorporate textual information tied to data instances to improve time series representations, typically through contrastive objectives that align text and time series embeddings in a shared semantic space.

CHARM takes a fundamentally different approach to incorporating textual information. Instead of relying on instance-level labels to build contrastive training pairs, CHARM leverages sensor-level descriptions as metadata. These textual embeddings are integrated directly into the featurization stage (via TCNs) or used to augment the self-attention mechanism. Rather than aligning modalities, CHARM learns mappings from textual metadata to guide representation learning. This design enables CHARM to scale to massive datasets where instance-level text labels are unavailable or impractical, requiring only sensor descriptions to improve the quality of learned time series representations.

Model	Multivariate	Channel Mixing	Invariance	Foundational	Channel Aware
Tloss	✓	✓	✗	✗	✗
TS2Vec	✓	✓	✗	✗	✗
TNC	✓	✓	✗	✗	✗
Autoformer	✓	✓	✗	✗	✗
FEDformer	✓	✓	✗	✗	✗
PatchTST	✓	✗	✗	✗	✗
CrossFormer	✓	✓	✗	✗	✗
iTransformer	✓	✓	✓	✓	✗
UniTS	✓	✓	✓	✓	✗
TimesFM	✗	–	–	✓	✗
MOIRAI	✓	✓	✓	✓	✗
MOMENT	✗	–	✓	✓	✗
TREP	✓	✓	✗	✗	✗
TOTO	✓	✓	✓	✓	✗
TimeMixer++	✓	✓	✗	✗	✗
CHARM	✓	✓	✓	✓	✓

Table 5: **a) Multivariate:** Can handle multivariate data[1]
b) Channel Mixing: Architecture enables learnable cross-channel interactions[2]
c) Invariance: Permuting the channels by a perturbation P ensures the outputs are also identically permuted.
d) Foundational: Can flexibly accept data of any arbitrary number of channels or time window.
e) Channel Aware: Uses sensor information to learn better representations.

B NOTATION

We denote matrices and tensors using boldface capital letters (e.g., $\mathbf{T}$, $\mathbf{E}$), and adopt *NumPy*-style indexing and slicing notation. Functions and operators are also denoted by bold capital letters, but are subscripted with θ to indicate parameterization, e.g., $\mathbf{E}_\theta$. The parameters θ may be learnable or fixed, depending on context. We reserve, $\mathbf{W}$ to represent the learnable weights in different layers of our architecture. An instance of a time series is represented as a tuple $\mathbf{t} = (\mathbf{T}, \mathbf{D}, \mathbf{pos})$. The first

[1]Multivariate here simply refers to whether a model can ingest multiple input channels, i.e. whether it can feasibly operate on a $T \times C$ data input, where $C > 1$. This is independent of whether the model is able to learn channel interactions, which is explicitly outlined in the channel mixing column.

[2]We do not consider models that are fundamentally univariate, but perform late fusion of channels at the representation level (by pooling for example), to be capable of channel mixing.

component, $\mathbf{T} \in \mathbb{R}^{T \times C}$, is a matrix of time series measurements, where T denotes the number of time points and C the number of channels. Each column $\mathbf{T}[:, i]$ corresponds to the uni-variate time series from channel i. The second component, $\mathbf{D}$, is an ordered list of length C, where each entry $\mathbf{D}[i]$ is a textual description of channel i, typically represented as a sentence or short passage. We assume that the descriptions in $\mathbf{D}$ are aligned with the corresponding columns of $\mathbf{T}$. The third component, $\mathbf{pos}$, represents the positional indices associated with the time series. We assume $\mathbf{pos} \in \mathbb{I}_+^T$ such that $|\mathbf{pos}| = T$. If not explicitly provided, we default to $\mathbf{pos} = [0, 1, \ldots, T-1]$. We denote the maximum time window size considered in our framework as $T_{\max}$.

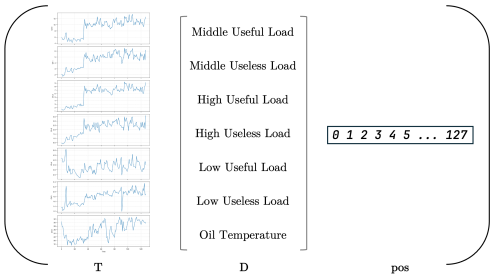


Figure 2: **Visualized representation of our data structure.** Note, that $\mathbf{T} \in \mathbb{R}^{T \times C}$, $|\mathbf{D}| = C$, and $\mathbf{pos} \in \mathbb{R}^T$

C Implementation Details

We attempt to follow the general set of best practices developed in the field of self-supervised learning, specifically those applicable to the Self-Distillation (Balestriero et al., 2023) family of algorithms. We outline the key details here;

1. **Optimization Schedule** We use an AdamW optimizer to optimize our model. The learning rate follows a linear warmup followed by a cosine decay.
2. **Weight Initialization** We use a fixed $\mathcal{N}(0, 0.02)$ initialization which is commonly used in pretraining large transformer models (OLMo et al., 2025).
3. **Weight Decay Scheduling** We use a cosine schedule for increasing the optimizer's weight decay over the course of training which has been shown to be crucial for training stability.
4. **EMA Schedule for Target Encoder** We use an exponentially moving average with a momentum schedule that is increased gradually over the course of training.

The weight decay scheduling and EMA schedule are identical to IJEPA (Assran et al., 2023). Besides sweeping over a few learning rates, we perform no additional hyperparameter tuning on the rest of the hyperparameters due to limited compute, and list them in Table 6.

C.1 Rotary Position Embeddings

Rotary Position Embeddings (RoPE) (Su et al., 2024) differ from traditional additive positional embeddings in that they encode positional information by rotating the query and key vectors in a structured, position-dependent manner. Unlike fixed or learned additive embeddings, RoPE is applied at **each** layer of the self-attention computation, allowing the model to encode relative position information directly into the attention mechanism.

Let $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{B \times T \times D}$ denote the queries and keys, where B is the batch size, T is the sequence length, and D is the hidden dimension. After linear projection and splitting into H attention heads:

$$\mathbf{Q}_h, \mathbf{K}_h \in \mathbb{R}^{B \times T \times d}, \quad \text{with } d = D/H$$

RoPE applies a deterministic rotation to each head's query and key vectors. For each position t and dimension index i, the rotation is defined as:

$$\text{RoPE}(\mathbf{x}_t)[2i] = \mathbf{x}_t[2i]\cos(\theta_{t,i}) + \mathbf{x}_t[2i+1]\sin(\theta_{t,i}) \tag{1}$$

$$\text{RoPE}(\mathbf{x}_t)[2i+1] = -\mathbf{x}_t[2i]\sin(\theta_{t,i}) + \mathbf{x}_t[2i+1]\cos(\theta_{t,i}) \tag{2}$$

$$\theta_{t,i} = t \cdot \omega_i, \quad \omega_i = 10000^{-2i/d} \tag{3}$$

where $\mathbf{x}_t$ denotes the tth token's vector (query or key), and ω_i are predefined inverse frequency terms.

In our implementation, we operate on inputs of shape $\mathbf{X} \in \mathbb{R}^{B\times T\times C\times d}$, where C represents the number of channels or sensors. To apply RoPE consistently across all channels, we broadcast the position encodings across the channel axis:

$$\widetilde{\mathbf{P}}_{b,t,c,:} = \mathbf{P}_{t,:}, \quad \forall\, b \in [1,B],\ c \in [1,C],\ t \in [1,T] \tag{4}$$

or equivalently, using broadcasting semantics:

$$\widetilde{\mathbf{P}} = \mathbf{P}[t,:] \longrightarrow \mathbb{R}^{B\times T\times C\times d} \tag{5}$$

This results in a broadcasted position encoding tensor $\widetilde{\mathbf{P}}$ where the same temporal position vector $\mathbf{P}_{t,:}$ is shared across all channels at time t, effectively associating the same position ID to multiple sensor tokens that occur at the same timestep.

C.2 Text Convolution Layer

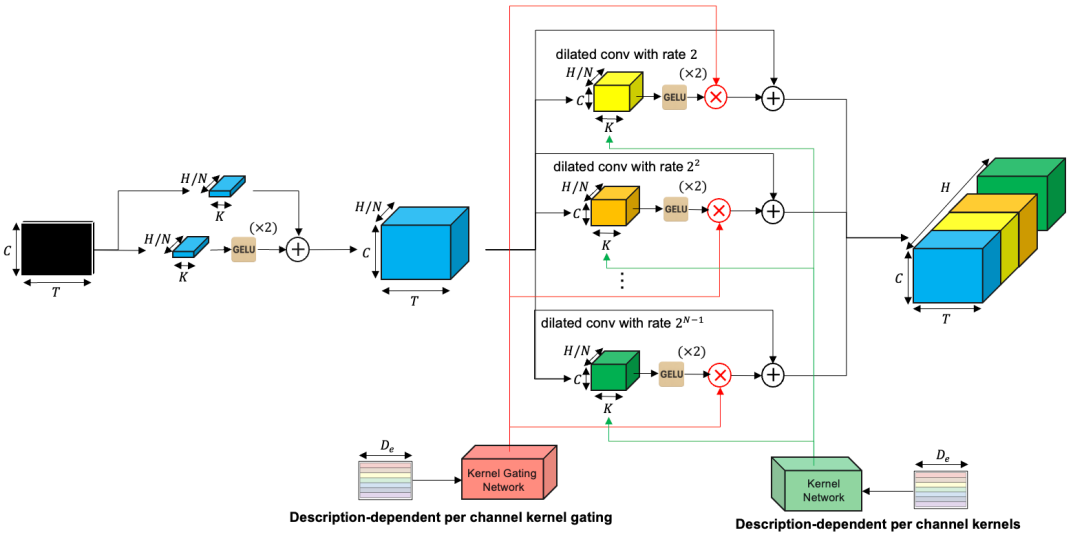


Figure 3: Schematic of the context-aware temporal convolutional network, performing initial featurization of multivariate time series inputs guided by granular textual descriptions of each channel.

C.2.1 Contrast to other featurization methods

Unlike typical TCNs, we concatenate activations across all intermediate layers to form a rich initial representation, see Figure 3. Our contextual TCN layer early in our model architecture closely relates to the concept of patching. Several recent foundation models for time series create non-overlapping static patches and project them through a single linear layer, e.g., (Das et al., 2024; Nie et al., 2023a; Woo et al., 2024b). These approaches can be generalized by interpreting convolution kernels as learnable linear mappings applied to strided segments of the data. Thus, our TCN layer represents a generalized, channel-aware extension of the patching concept.

C.2.2 Implementation

To compute convolutions efficiently across all sensors and batches, we stack the convolutional kernels corresponding to each sensor description and reshape the input to treat the $B \times C \times H$ channels as independent time series. We then apply a grouped 1D convolution using `F.conv1d` with $B \times C \times H$ groups, where each element in the original $[B, T, C, H]$ input is treated as a separate time series along the time axis. This allows us to apply distinct filters for each batch, channel, and embedding dimension in parallel.

C.2.3 Initialization

Despite the effectiveness of this mechanism, careful numerical stabilization of the convolution kernels is essential. To achieve this, we first apply a non-parametric LayerNorm to z-normalize the sensor embeddings, $\mathbf{E}_d$. The projection matrix within the kernel network is then initialized using Xavier normal initialization (Glorot & Bengio, 2010). Subsequently, we re-normalize the resulting kernels $\mathbf{W}_k$ as

$$\mathbf{W}_k = \textbf{LayerNorm}(\mathbf{W}_k) \cdot \sqrt{\frac{2}{K}}$$

Since our TCN layer employs GeLU nonlinearities, this initialization approach aligns with Kaiming initialization principles, (He et al., 2015), and ensures stable activations, preventing them from progressively exploding or vanishing across convolution layers.

C.3 Model Sizing

For the given hyperparameter set $N = 8, d = 128, \text{ff}_{\text{dim}} = 4d$, our pretrained model is ~7.1M parameters.

C.4 Contextual Attention Layer Details

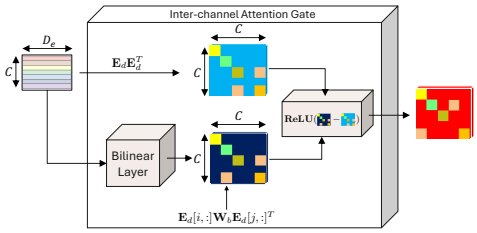


Figure 4: Description-aware gating mechanism that selectively suppresses cross-channel attention.

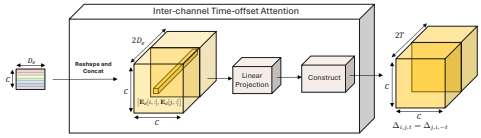


Figure 5: Symmetric inter-channel temporal-offset attention encoding dependencies across time lags.

C.5 Additional Modifications to the Transformer Layers

In line with recent developments in large scale pretraining of transformer based architectures, we implement several modifications that diverge from the original transformer architecture.

SwiGLU We replace the regular feedforward layers with a SwiGLU feedforward layer.

QK-norm We add a pre-attention layernorm to the queries and keys.

Rotary Position Embeddings Instead of using sinusoidal positional embeddings, we use rotary positional embeddings which are applied on the queries and keys at every layer. The positional indices are provided through the **pos** argument.

Reordering Sublayers We experiment with using 3 approaches to assess the optimal configuration of the transformer sublayers.

$$\begin{cases} x = \text{norm}(x + \text{SubLayer}(x)) & \text{Post Norm} \\ x = x + \text{SubLayer}(\text{norm}(x)) & \text{Pre Norm} \\ x = x + \text{norm}(\text{SubLayer}(x)) & \text{Swin Norm} \end{cases} \tag{6}$$

In the case of Pre Norm and Swin Norm, we also experiment with adding LayerNorms in the main transformer branch every n layers, to ensure further stability.

C.6 Efficient Computation Techniques

C.6.1 Slice And Tile Attention Layers

To vectorize the process of generating the full $\bar{\mathbf{\Delta}}$ tensor, we provide the pytorch pseudocode versions of the naive and vectorized versions in Figure 6 and Figure 7.

Figure 6: Naïve attention-weight matrix construction

```python
def build_attention_weight_matrix(time_deltas: Tensor,
                                  T_proj: Tensor) -> Tensor:
    """
    Constructs the full attention weight matrix by explicit loops.
    Args:
        time_deltas: LongTensor, shape (T, T)
        T_proj:      Tensor, shape (B, C, C, 2*T - 1)
    Returns:
        attn:        Tensor, shape (B, C*T, C*T)
    """
    B, C, _, T1 = T_proj.shape
    T = time_deltas.size(0)
    assert 2 * T - 1 == T1

    attn = torch.zeros((B, C * T, C * T), device=T_proj.device)
    for i in range(T):
        for j in range(T):
            delta = time_deltas[i, j].item()
            block = T_proj[..., delta]    # (B, C, C)
            attn[..., i*C:(i+1)*C, j*C:(j+1)*C] = block
    return attn
```

D Training Loss

D.1 Self-supervised Loss

We employ a multi-resolution ℓ_1 loss that aligns predictor and target embeddings at three levels of granularity. Let $\bar{\mathbf{X}}^t = \mathbf{X}^t[\mathcal{I}_{\text{prd}}, :] \in \mathbb{R}^{|\mathcal{I}_{\text{prd}}| \times C \times H}$ and $\bar{\mathbf{X}}^p = \mathbf{X}^p[-|\mathcal{I}_{\text{prd}}| :, :] \in \mathbb{R}^{|\mathcal{I}_{\text{prd}}| \times C \times H}$. The self-supervised loss is defined as

$$\mathcal{L}_{\text{ssl}} = \sum_{i,j,t} \left|\bar{\mathbf{X}}^p_{i,j,t} - \bar{\mathbf{X}}^t_{i,j,t}\right| + \sum_{i,t} \left|\mu_j\left(\bar{\mathbf{X}}^p_{i,:,t}\right) - \mu_j\left(\bar{\mathbf{X}}^t_{i,:,t}\right)\right| + \sum_{t} \left|\mu_{i,j}\left(\bar{\mathbf{X}}^p_{:,:,t}\right) - \mu_{i,j}\left(\bar{\mathbf{X}}^t_{:,:,t}\right)\right|$$

with $\mu_j(\mathbf{X}_{i,:,t}) = \frac{1}{C}\sum_j \mathbf{X}_{i,j,t}$ and $\mu_{i,j}(\mathbf{X}_{:,:,t}) = \frac{1}{CT}\sum_i \sum_j \mathbf{X}_{i,j,t}$. This multi-resolution loss encourages the model to align representations both at the fine-grained level (per time point and channel) and at higher levels of abstraction (per time point and globally).

D.2 Regularization Loss

We include two regularization terms related to key modules of the contextual attention layer, namely the inter-channel gating and inter-channel time-offset attention modules. Given the inherent sparsity

Figure 7: Fast attention-weight matrix construction

```python
def build_attention_weight_matrix_fast(time_deltas: Tensor,
                                       T_proj: Tensor) -> Tensor:
    """
    Block-wise assembly via tensor indexing and reshape.
    """
    B, C, _, T1 = T_proj.shape
    T = time_deltas.size(0)
    assert 2 * T - 1 == T1

    # 1) Flatten index grid
    flat_idx = time_deltas.view(-1)        # shape (T*T,)

    # 2) Gather all needed projection slices at once
    gathered = T_proj.index_select(dim=-1, index=flat_idx)
    #    result: (B, C, C, T*T)

    # 3) Reshape to (B, C, C, T, T)
    gathered = gathered.view(B, C, C, T, T)

    # 4) Reorder to (B, T, C, T, C)
    gathered = gathered.permute(0, 3, 1, 4, 2)

    # 5) Collapse blocks into (B, C*T, C*T)
    return gathered.contiguous().view(B, C * T, C * T)
```

in meaningful channel relationships, we promote sparsity in the learned channel relationships, by regularizing the similarity threshold matrix $\mathbf{Z}$ toward 1 and regularizing the relationships among channels across temporal offsets using

$$R_1 = \sum_{i,j} |1 - \mathbf{Z}[i,j]|, \quad R_2 = \frac{\sum_{i,j} \sum_t \mathbf{\Delta}[i,j,t]^2}{C^2},$$

respectively. The regularization term R_2 encourages consistency and stability in the learned inter-channel temporal relationships. Combining these loss terms results in the following training objective function to be applied across all data points $\mathcal{L} = \mathcal{L}_{\text{ssl}} + \lambda_1 R_1 + \lambda_2 R_2$, where λ_1 and λ_2 control the extent and strictness of gating and temporal attention suppression.

E JEPA

E.1 Dataset Generation

The core principle of JEPA-based self-superived training involves producing representations for two augmented views originating from the same data instance. JEPA training aims to minimize a discrepancy measure (e.g., ℓ_1 or ℓ_2) between these representations. In vision, these views commonly result from image augmentations such as jittering, masking, or cropping.

Figure 9 presents a visual representation of our JEPA tasks, which rely on learning 1) causal representations and 2) smoothing representations.

E.2 JEPA Encoders – Deep Dive

In this section we dive a bit deeper into our implementation of the JEPA framework. We denote the TCN layer that featurizes our input time series as $\mathbf{F}$ and our encoder stack (of N layers), as $\mathbf{E}$.

As outlined earlier, our featurizing layer converts a multivariate time series instance to an embedded version of the time series with the same leading dimensions, i.e.;

$$\mathbf{F} : \mathbb{R}^{T \times C} \to \mathbb{R}^{T \times C \times H}$$

Category	Hyperparameter	Value
Optimization Schedule	Optimizer	AdamW
	ϵ	1e-8
	β_1	0.95
	β_2	0.99
	epochs	100
	gradient clipping	2.0
	λ_1, λ_2	1e-5
	batch size [2]	4
	gradient accumulation	2
Scheduler	starting LR	1e-8
	final LR	1e-6
	starting weight decay	0.04
	final weight decay	0.4
	learning rate schedule	linear warmup $\rightarrow$ cosine decay
	weight decay schedule	cosine
	fraction of warmup epochs	0.1
	scale factor [3]	1.25
Data	window size	512
	stride	128
	minimum samples per dataset	400
	maximum samples per dataset	1000
SSL Task Parameters	number of targets	4
	C_{min}	0.3
	C_{max}	0.4
	T_{min}	0.1
	T_{max}	0.2
JEPA Architecture	encoder layers	8
	predictor layers	4
	encoder dim	128
	predictor dim	64
Model Architecture	feedforward layer	SwiGLU
	ff_dim_multiplier	4
	attention dropout	0.01
	norm	non-parametric layernorm
	attention configuration	pre-norm

Table 6: Hyperparameters for full training pipeline

On the other hand our encoder ingests the embedded time series and returns a contextually embedded time series while maintaining the same output dimensions i.e.;

$$\mathbf{E} : \mathbb{R}^{T \times C \times H} \rightarrow \mathbb{R}^{T \times C \times H}$$

Given this notation, our 3 JEPA networks (Context, Target, Predictor) can be represented as:

$$\textbf{Context} \Rightarrow [\mathbf{F} \rightarrow \mathbf{E_1}] \tag{7}$$

$$\textbf{Target} \Rightarrow [\mathbf{F} \rightarrow \mathbf{E_1}] \tag{8}$$

$$\textbf{Predictor} \Rightarrow [\textbf{DownProj} \rightarrow \mathbf{E_2} \rightarrow \textbf{UpProj}] \tag{9}$$

Now, with this featurization and encoder layer stack, we provide a PyTorch style pseudocode of the JEPA framework, i.e. the data flow between the Context, Target, and Predictor encoders in Section E.2, Section E.2, and Section E.2.

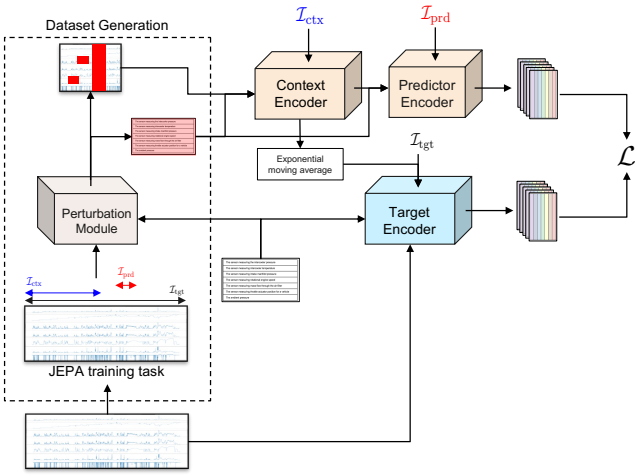


Figure 8: JEPA architecture with three encoders processing augmented views.

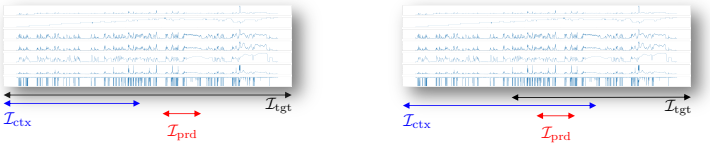


Figure 9: JEPA Tasks Visualized : Causal Prediction (left) Smoothing (right)

```python
class ContextTgtEncoder:
    def forward(self, x, ctx_idx):
        """
        x : [..., T, C]
        """
        x = self.featurizer(x)
        for layer in self.encoder_layers:
            x = layer(x, ctx_idx)
        return x
```

Figure 10: Context and Target Network

```python
class Predictor:
    def forward(self, ctx_embeds, ctx_idx, target_idx):
        """
        embeds : [..., T, C, H]
        target_pos : [..., T2]
        """
        x = self.downproj(ctx_embeds)  # [..., T, C, H1]
        mask_tokens = broadcast(self.mask_token, target_idx)  # [..., T2, C, H1]

        x = concat([x, mask_tokens])  # [..., T+T2, C, H1]
        pos = concat([ctx_idx, target_idx])

        for layer in self.encoder_layers:
            x = layer(x, pos)
        x = self.upproj(x)  # [..., T+T2, C, H]
        x = x[..., -target_idx.size(-2):, :, :]  # [..., T2, C, H]
        return x
```

Figure 11: Predictor Network

```python
class JEPA:
    def __init__(self):
        self.context_encoder = ContextTgtEncoder()
        self.target_encoder = copy_and_freeze_params(self.context_encoder)
        self.predictor = Predictor()

    def forward(self, x, ctx_idx, tgt_idx):
        """
        x : [..., T, C]
        """

        # get full embeddings
        full_embeds = self.target_encoder(x)

        # get context embeddings
        context_embeds = self.context_encoder(x[..., ctx_idx, :])

        # get predicted embeddings
        predicted_embeds = self.predictor(context_embeds, ctx_idx, tgt_idx)
        target_embeds = full_embeds[..., tgt_idx, :, :]

        # compute loss
        loss = loss_fn(predicted_embeds, target_embeds)
        return loss
```

Figure 12: JEPA

F HARDWARE

We use a cluster of 8 80GB NVIDIA A100 GPUs. We use Distributed Data Parallelism to speed up training, along with bf-16 mixed precision. Our models are implemented in PyTorch (Paszke et al., 2019), and training is done with PyTorch lightning (Falcon & The PyTorch Lightning team, 2019). We handle our configuration management using gin configs.

G LIMITATIONS

The primary limitations of our model are:

1. **Limited context lengths**
   Given our model's architecture, we are required to compute attention scores over the entire $C \times T$ input. As we do not rely on downsampling/patching, we compute the full $\mathcal{O}(C^2T^2)$ attention matrix, which can be prohibitively large, especially for datasets with a large number of unrelated channels, or extremely long time horizons. Potential workarounds to this are computing the attention scores for only relevant channel pairs, based on pre-filtering similar channels based on the channel gating scores (high gating scores effectively clamp the attention scores completely, and self attention between these channels is effectively wasted compute). For large time horizons, a downsampling/patching layer can be appended to the encoder stack prior to the TCNs to operate on a lower effective time window.

2. **Sensitivity to missing or noisy metadata**
   CHARM conditions its featurization and attention layers on channel descriptions, raising a natural question about robustness when metadata is missing, noisy, or uninformative. Our ablations (Section L) provide a detailed picture of graceful degradation. Replacing curated descriptions with ordinal placeholders ("Sensor1", "Sensor2") costs only 1.1 pp in classification accuracy (70.3% vs. 71.4%, Table 26). Replacing text embeddings with fixed per-channel-index vectors—removing the text modality entirely while preserving the conditioning architecture—yields 68.3% classification accuracy (Table 29), confirming that the conditioning mechanism itself carries most of the benefit. Substituting random embeddings (same parameter count, zero semantic content) degrades forecasting MSE by 0.9%, while shuffling descriptions across channels (correct distribution, wrong correspondence) degrades classification by 2.6 pp—worse than random—demonstrating that the model actively exploits correct channel–text mappings rather than merely benefiting from added capacity.

   In practice, channel metadata is commonly available in real-world sensor deployments (column headers, SCADA tags, data sheets, FHIR fields). For the UEA benchmark, which provides dataset-level documentation but not per-channel labels, we manually curated descriptions—a time-consuming effort that does not scale. We have since found that a single LLM call per dataset can generate adequate descriptions from column headers and metadata, substantially reducing this overhead. Nevertheless, the reliance on channel metadata remains a practical constraint, and developing mechanisms that adaptively route the model based on metadata availability is an important direction for future work.

3. **Robustness to strong domain shifts**
   We evaluated CHARM on datasets entirely excluded from pretraining to test generalization: the Exchange Rate forecasting benchmark (8 channels, financial domain) and the UCI Hydraulic Systems dataset (17 heterogeneous sensors at mixed sampling rates of 100 Hz, 10 Hz, and 1 Hz). On Exchange Rate, frozen CHARM embeddings with a linear head remain competitive with fully supervised baselines, outperforming iTransformer at horizon 336. On UCI Hydraulics, CHARM achieves 99.8% valve-condition classification accuracy, exceeding published supervised baselines by $\sim$4 pp with no task-specific training. These results demonstrate meaningful transfer across domains and sampling rates.

   However, we have not systematically evaluated robustness under more extreme distributional shifts, such as entirely novel sensor modalities (e.g., radar, LiDAR) or radically different temporal resolutions (sub-millisecond or multi-year). In such regimes, the pretraining distribution may provide insufficient coverage of relevant temporal patterns. We also note that channel-index vectors (Table 29) are inherently dataset-specific and cannot transfer to unseen sensor configurations, whereas text embeddings carry portable semantics that enable cross-dataset transfer—a property exercised in our held-out evaluations. Characterizing the boundary conditions of this transfer capability, and developing strategies for few-shot adaptation under strong domain shift, is an important direction for future work.

H MODEL COMPLEXITY

Our pretraining was conducted on **8 A100 GPUs** over approximately **18 hours**, inclusive of minor overheads for dataset preprocessing, downstream evaluations, and logging. Training was performed with **bf16 mixed precision** under **distributed data parallelism**. A frozen text embedding model was invoked during training, served independently on a single L4 GPU. The **peak GPU memory usage** per device was **72.7 GB**, while **peak CPU utilization** remained at **4 GB**.

H.1 ARCHITECTURAL CONTRIBUTIONS TO COMPLEXITY

Relative to conventional transformer architectures, the primary increase in model complexity arises from the inclusion of the **temporal convolutional (TCN) layer** and **text-attention layers**. For a representative configuration—where the text embedding dimension is 384, the time-series embedding dimension is 256, and the convolution kernel size is 8—the parameter counts for the additional modules are enumerated in Table 7.

Layer	Count	Number of Parameters
TCN	$\mathbf{D}_{\text{text}} \times H \times K$	$384 \times 256 \times 8 = 786{,}432$
Gating Layer	H^2	$256^2 = 65{,}536$
Time-Delta Layer	$2 \times \mathbf{D}_{\text{text}} \times T_{\max}$	$2 \times 384 \times 1500 = 1{,}152{,}000$

Table 7: Parameter counts for the TCN and text-attention modules.

In total, these components introduce approximately **2 million additional parameters**, corresponding to $\sim$25% of the overall model size.

ENCODER COMPOSITION

The pretraining framework employs **three modules**: a **context encoder**, a **target encoder**, and a **predictor**. The target encoder parameters are non-trainable and are tied to the context encoder parameters but consume GPU memory equivalent to the context encoder during forward passes. The predictor is comparatively lightweight, operating at lower dimensionality with fewer layers.

The **parameter distribution** across modules is as follows:

- Context encoder: **7.1M parameters**
- Target encoder: **7.1M parameters (frozen)**
- Predictor: $\sim$**4M parameters**

The combined model therefore comprises $\sim$**18.2M parameters**, of which $\sim$**11.1M are trainable**. For computing the embeddings at inference we only utilize the context encoder, which means the embeddings are the output of a **7.1M** parameters model.

I DATASETS

Here we provide a list of dataset sources we used to train our model. Wherever sensor names were not readily available, we manually curate the sensor descriptions from the dataset specifications.

UEA Dataset The UEA Dataset is a popular publicly available dataset used for benchmarking time series classification algorithms. We restrict ourselves to a subset of the full 30 datasets, as not all of them have meaningful sensor descriptions. For a few of the datasets within UEA, we manually annotate the descriptions based on the official paper (Bagnall et al., 2018).

Liu-ICE Machine Fault Dataset The Liu-ICE Machine Fault Dataset is a real world fault diagnosis dataset which consists of data collected from an internal combustion engine test bench. The dataset consists of multiple different kinds of fault scenarios, and comes with a publicly available benchmark.

Electricity Transformer Dataset The Electricity Transformer Dataset (ETTDataset/ETDataset) is a widely used dataset for time series forecasting, which contains data of dynamic power loads in an electric power grid located in China. This dataset contains 4 sub-datasets (`ETTh1`, `ETTh2`, `ETTm1`, `ETTm2`), which operate at different granularities.

Weather The Weather Dataset from the MPI is a real world dataset of meteorological indicators for the year of 2020.

Electricity The Electricity dataset contains hourly consumption from multiple consumers from 2012 to 2014.

Illness The Illness Dataset includes weekly records for patients suffering from influenza like illnesses collected by the CDC.

SKAB - Skoltech Anomaly Benchmark Dataset The SKAB dataset is designed for evaluating anomaly detection, targeted at two main problems : outlier detection and changepoint detection

Gas Sensor Array Modulation The Gas Sensor Array Modulation from the UCI Machine Learning Repository is collection of time-series recordings obtained from an array of metal-oxide gas sensors.

Machinery Fault Dataset The Machinery Fault Dataset comprises six different simulated states: normal function, imbalance fault, horizontal and vertical misalignment faults and, inner and outer bearing faults from a machinery fault simulator.

Metro PT-3 Dataset The MetroPT-3 dataset is a multivariate time series collection created for predictive maintenance in the railway industry. It consists of over 1.5 million records (instances) captured at 1Hz from a train compressor's Air Production Unit (APU) over the period from February to August 2020.

Unleashing the Power of Wearables The Human Activity Recognition Trondheim (HEART) dataset is a professionally annotated collection designed for developing machine learning algorithms capable of recognizing human activities in a free-living environment. Created at the Norwegian University of Science and Technology (NTNU), it features 22 subjects who wore two 3-axis Axivity AX3 accelerometers for approximately 2 hours each while performing various daily tasks. The sensors were placed on the right thigh and lower back, providing multivariate time series data sampled at 50Hz.

Predictive Maintenance of Hydraulic Systems The Predictive Maintenance of Hydraulic Systems dataset contains multivariate time series data collected from a hydraulic test rig. This dataset includes sensor readings—such as pressures, volume flows, temperatures, and more—recorded during load cycles of the hydraulic system.

We provide a summary of the specifications of each dataset in Table 8. If a dataset is present in a downstream benchmark, we only include the defined "train" subset of the full dataset, to prevent the model from optimizing an SSL loss over the test dataset samples.

J Data Loading

To enable efficient data loading, we perform under/over sampling to balance the datasets. The degree of under/over sampling is controlled by the t_1 : `min_samples_per_dataset` and t_2 : `max_samples_per_dataset` parameters, which upsamples or downsamples the data if the number of samples is either $< t_1$ or $> t_2$ respectively.

Following this, each dataset is handled by its own dataloader, which cyclically yields batches of data from each dataset at every training step. This is handled internally by `pytorch lightning`'s

[4]The UEA Datasets are provided as windowed instances, i.e. they are not hosted as contiguous, chronological blocks of shape $T \times C$, but rather stored as $N \times T' \times C$. Here, we compute the "# of timesteps" as $N \times T'$, although there may be redundant overlaps based on how the data was collected and labelled.

Dataset Name	#Timestamps	#Channels
Open-Source/Kaggle Datasets		
Appliances Energy Prediction	19,735	26
Gas Sensor Array Temperature Modulation	3,843,160	19
Household Electric Power Consumption	2,075,259	7
Machinery Fault Diagnosis	487,748,049	8
MetroPT-3 Dataset	1,516,948	15
Predictive Maintenance of Hydraulics System	132,300	17
SKAB - Skoltech Anomaly Benchmark	46,860	8
Unleashing the Power of Wearables	6,461,328	6
Liu	288,623	10
UEA Datasets [4]		
NATOPS	9180	24
Epilepsy	28222	3
Articulary Word Recognition	39600	9
UWave Gesture Library	37800	3
Cricket	129276	6
ERing	1950	4
Character Trajectories	169218	3
Finger Movements	15800	28
SelfRegulation SCP1	240128	6
Basic Motions	4000	6
Atrial Fibrillation	9600	2
Hand Movement Direction	64000	10
Handwriting	22800	3
Libras	8100	2
LSST	88,524	6
Racket Sports	4530	6
Forecasting Benchmark Datasets		
`ETTh1`	17,420	7
`ETTh2`	17,420	7
`ETTm1`	69,680	7
`ETTm2`	69,680	7
Weather	52,696	21
Illness	966	7

Table 8: Overview of datasets categorized into Open-Source/Kaggle, UEA, and Forecasting benchmark datasets.

`CombinedLoader` method, which yields a batch from each dataloader (if the iterator is not yet exhausted). As a result, our `effective batch size` [5]per step is now computed as :

$$(\texttt{batch size}) \times (\text{\# of GPUs}) \times (\text{\# of datasets}) \times (\texttt{grad_accum_steps}) \tag{10}$$

At the beginning of every epoch, we reload all datasets, which results in fresh under/over sampling indices. This enables the support of larger datasets to be incrementally covered over multiple epochs of training.

The JEPA tasks are randomly sampled after the datasets are sampled, which results in fresh context and target masks for repeated samples. This avoids the exact same sample being repeated several times in an epoch for underrepresented datasets, due to stochasticity in how the masks are generated.

[5]The "# of datasets" technically refers to the number of unexhausted datasets on that training step, as each dataset has a different number of samples.

J.1 PERTURBATIONS

Our augmentation design is directly motivated by failure modes frequently observed in real-world time-series data—particularly in industrial and sensor-driven applications— where channel- or block-level gaps occur due to intermittent sensor outages, network disruptions, or scheduled maintenance. To build robustness against such artifacts, we incorporate two principled time-domain masking strategies:

- **Uniform segment masking:** masks a contiguous temporal segment across all channels, simulating system-level events such as edge-cache dropout or network-wide packet loss.
- **Channel-selective masking:** applies the same temporal mask to a randomly selected subset of channels, capturing sensor-specific anomalies such as probe failure or drifting instrumentation.

These perturbations are applied solely to the context encoder's input during training, while the teacher view remains unperturbed. This asymmetry forces the model to leverage broader temporal and cross-channel structure for representation learning, in line with the JEPA framework's core principle of predicting masked target representations rather than raw values.

The augmentation functions are tailored to the time-series domain but echo proven techniques in analogous modalities. For instance, our segment masking is a temporal analogue to SpecAugment's `TimeMasking`, a canonical augmentation for large-scale speech models (implemented in `torchaudio.transforms.TimeMasking`). Similar masking-based augmentations have also been adopted in recent time-series representation learning methods such as TEST (Sun et al., 2024).

Hyperparameters controlling mask width and frequency were chosen based on prior experience with industrial time-series systems. Due to computational budget constraints, we did not conduct a systematic hyperparameter sweep, instead prioritizing augmentations with clear interpretability and real-world grounding.

K EXPERIMENTS

K.1 CLASSIFICATION

K.1.1 UEA CLASSIFICATION BENCHMARK

Dataset We evaluate our model on the popular UEA Dataset which serves as a standard time series classification benchmark for multivariate data. We consider a subset of 21 UEA datasets (list in Table 10) that cover a diverse set of tasks and domains. We select these datasets on the basis of what our model's default context length can handle in a single GPU. As we modify the attention mechanism directly, we cannot leverage existing efficient implementations, and thus are restricted by a maximum context window size. Formally, we select the subsets based on the following rule: num channels < 50, num timestamps < 1500.

Tasks	Supervision	Datasets	# datasets	Metrics	Baselines
Classification	Frozen SVM; Fine-tuned+Linear	UEA	21	Accuracy, Wins, Total Correct	MiniROCKET, TS2Vec, T-Loss, TS-TCC, T-Rep, MOMENT
Anomaly Detection	Frozen reconstructor (linear head)	UCR-AD	46	Adjusted Best F1	Anomaly Transformer, DGHL, GPT4TS, TimesNet, MOMENT (0, LP), TS2Vec, T-Rep
Anomaly Detection	Frozen reconstructor (linear head)	SKAB	34	F1, FAR, MAR	T^2, T^2+Q (PCA), Isolation Forest, MSET, Feed-Forward AE, Conv-AE, LSTM-AE, VAE, LSTM-VAE, MSCRED, TS2Vec, T-Rep, MOMENT
Long-horizon Forecasting	Per-dataset linear probe (frozen); universal non-linear head (frozen encoder); universal non-linear head (unfrozen encoder)	ETTh1 ETTh2 ETTm1 ETTm2 Weather Exchange Rate Illness	7	MSE, MAE	T-Rep, TS2Vec, PatchTST, MOMENT, Toto, TIMEMIXER++, Moirai, VisionTS

Table 9: Unified baseline summary by task, now including dataset counts.

Task Description Given a labeled time series data instance (X, y), where X is a multivariate time series, and y corresponds to a supervised label corresponding to X, our goal is to learn a classifier h to minimize test error, i.e. $\epsilon = \mathbb{E}_{(x,y)\sim\mathbb{P}_{(x,y)}}[1(h(x) \neq y)]$.

Downstream Setup We evaluate the quality of our representations, $Z \in \mathbb{R}^{B\times T\times C\times H}$ in the following setups.

1. **Frozen + off-the-shelf non-linear classifier (SVM)**
   Similar to Goswami et al. (2024), we flatten our embeddings, Z and feed them to an SVM with the standard set of hyperparameters proposed in Franceschi et al. (2019), which are also used in `T-Rep`, `TS2Vec`, `T-Loss`, etc. The hyperparameters are chosen for each dataset separately, using 5-fold cross validation on the train set.
2. **Finetuned + linear probe**
   Similar to the linear probing setup in Goswami et al. (2024), we finetune the encoder for

	Dataset	Ch.	Len.	Inc.
	ArticularyWordRecognition	9	144	✓
	AtrialFibrillation	2	640	✓
	BasicMotions	6	100	✓
	CharacterTrajectories	3	182	✓
	Cricket	6	1,197	✓
	Epilepsy	3	206	✓
	ERing	4	65	✓
	FingerMovements	28	50	✓
	HandMovementDirection	10	400	✓
	Handwriting	3	152	✓
	JapaneseVowels	12	29	✓
	Libras	2	45	✓
	LSST	6	36	✓
	NATOPS	24	51	✓
r0.5	PenDigits	2	8	✓
	Phoneme	11	217	✓
	RacketSports	6	30	✓
	SelfRegulationSCP1	6	896	✓
	SelfRegulationSCP2	7	1,152	✓
	SpokenArabicDigits	13	93	✓
	UWaveGestureLibrary	3	315	✓
	DuckDuckGeese	1,345	270	×
	EigenWorms	6	17,984	×
	EthanolConcentration	3	1,751	×
	FaceDetection	144	62	×
	Heartbeat	61	405	×
	InsectWingbeat	200	78	×
	MotorImagery	64	3,000	×
	PEMS-SF	963	144	×
	StandWalkJump	4	2,500	×

Table 10: Overview of UEA datasets included in CHARM evaluation.

each dataset separately. We flatten the embeddings Z, and feed them to a single linear layer which maps the embeddings to a vector of logits, trained with a cross entropy loss. The training hyperparameters are listed in Table 11, which is the same for all UEA datasets.

Hyperparameter	**Value**
Batch size	16
Learning rate	$1e-4$
Weight decay	$1e-4$
Epochs	500
Optimizer	Adam
Label smoothing	0

Table 11: Training hyperparameters for finetuning setup

Baselines To position ourselves in the existing landscape of time series classification methods, we include baselines from the following set of approaches:

1. Time Series Classification Models: `MiniRocket`
2. Semantic Representation Learning Models : `T-Rep`, `TS2Vec`, `T-Loss`, `TS-TCC` etc.
3. Reconstruction Based Representation Learning Models : `MOMENT`

Given our limited compute availability, all baseline results reported in the results table are drawn from prior published work. We restrict our comparison to models with results on the majority of the UEA datasets, and therefore exclude models with incomplete or missing UEA coverage (e.g., `UniTS`).

Metrics To compare our model's performance on the combined set of UEA Datasets, we measure 3 quantities:

Average Accuracy. For dataset i with n_i samples:

$$\text{Acc}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{1}[h(x_{ij}) = y_{ij}] \,,$$

and the average accuracy across D datasets is

$$\text{AvgAcc} = \frac{1}{D} \sum_{i=1}^{D} \left(\frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{1}[h(x_{ij}) = y_{ij}] \right) .$$

Number of Correctly Classified Samples.

$$\text{NumCorrect} = \sum_{i=1}^{D} \sum_{j=1}^{n_i} \mathbf{1}[h(x_{ij}) = y_{ij}] \,.$$

Number of Wins. For M models $\{h_m\}_{m=1}^{M}$, define accuracy of model m on dataset i as

$$\text{Acc}_{i,m} = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{1}[h_m(x_{ij}) = y_{ij}] \,.$$

The number of wins for model m is

$$\text{Wins}(m) = \sum_{i=1}^{D} \mathbf{1} \Big[\text{Acc}_{i,m} = \max_{m'} \text{Acc}_{i,m'} \Big] \,.$$

We report average accuracy and number of wins as they are standard measures used in other papers that use the UEA benchmark, however, as noted in Fleming & Wallace (1986), we would like to highlight that relying on averages of arithmetic means in such setups might be misleading, as the number of test samples vary significantly per dataset (see Figure 13). As a result we additionally include unnormalized scores, which we empirically observe to be a relatively less noisier metric to track during training.

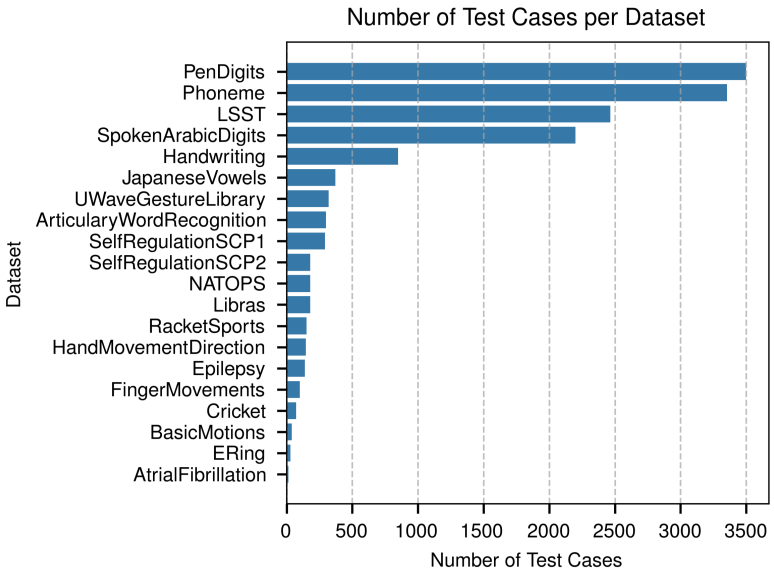


Figure 13: Sizes of different UEA Datasets

Results Results for CHARM under the frozen + SVM setup are presented in Table 12, while the finetuned version is reported in Table 14. Overall, CHARM demonstrates strong aggregate performance, particularly in terms of average accuracy and total correct predictions across datasets. Moreover, we observe competitive results on datasets excluded from pre-training (`JapaneseVowels`, `PhonemeSpectra`, `PenDigits`), highlighting the strong generalization ability of the learned embeddings. The substantial improvement from finetuning on individual datasets suggests that post-training strategies can be effectively used to adapt the model for classification tasks.

Dataset	**TS2Vec**	**T-Loss**	**TS-TCC**	**T-Rep**	**MOMENT**	**MiniROCKET**	**CHARM**$_{\text{frozen + SVM}}$
AtrialFibrillation	29	13	27	35	20	20	47
Articulary/WordRecognition	97	94	95	97	99	98	99
BasicMotions	100	100	100	100	100	100	98
CharacterTrajectories	99	99	99	99	—	99	98
Cricket	95	97	92	96	99	97	96
ERing	90	13	90	94	96	93	96
Epilepsy	96	97	96	97	99	100	98
FingerMovements	50	58	46	50	49	42	59
HandMovementDirection	42	35	24	54	32	41	54
Handwriting	46	45	50	41	31	24	33
LSST	56	51	47	53	41	67	60
Libras	86	88	82	83	85	94	83
NATOPS	90	92	82	80	83	92	82
RacketSports	89	86	82	88	80	88	86
SelfRegulationSCP1	79	84	82	82	84	88	82
UWaveGestureLibrary	88	88	75	89	91	91	91
SpokenArabicDigits	99	91	97	99	98	99	97
SelfRegulationSCP2	55	54	53	59	48	49	58
Japanese Vowels †	97	99	93	96	72	92	97
Phoneme Spectra †	24	22	25	23	23	28	20
Pen Digits †	98	98	97	97	97	97	98
Wins	2	5	2	3	3	**7**	4
Avg. Accuracy	76.4	71.6	73.1	76.8	71.3*	76.2	**77.6**
Total Correct	7171	6836	6835	7075	5204*	**7284**	7139

Table 12: Comparison of classification accuracy across multiple datasets and models. Datasets marked with † were not included in pre-training CHARM. *MOMENT does not report scores for `CharacterTrajectories`, and we exclude it while calculating MOMENT's scores.

Hyperparameter	Value
C	{0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000}
kernel	{'rbf'}
degree	{3}
gamma	{'scale'}
coef0	{0}
shrinking	{True}
probability	{False}
tol	{0.001}
cache_size	{200}
class_weight	{None}
verbose	{False}
max_iter	{10000000}
decision_function_shape	{'ovr'}
random_state	{None}

Table 13: SVM Hyperparameter Grid

Dataset	TS2Vec	T-Loss	TS-TCC	T-Rep	MOMENT	MiniROCKET	$\text{CHARM}_{\text{frozen+SVM}}$	$\text{CHARM}_{\text{finetune}}$
AtrialFibrillation	29	13	27	35	20	20	**47**	40
Articulary/WordRecognition	97	94	95	97	**99**	98	99	**99**
BasicMotions	**100**	**100**	**100**	**100**	**100**	**100**	98	**100**
CharacterTrajectories	99	**99**	99	99	–	99	98	99
Cricket	95	97	92	96	**99**	97	96	94
ERing	90	13	90	94	96	93	**96**	94
Epilepsy	96	97	96	97	99	**100**	98	99
FingerMovements	50	58	46	50	49	42	**59**	57
HandMovementDirection	42	35	24	54	32	40	**54**	51
Handwriting	46	45	**50**	41	31	24	33	36
LSST	56	51	47	53	41	67	60	**71**
Libras	86	88	82	83	85	**94**	83	87
NATOPS	90	92	82	80	83	92	82	**92**
RacketSports	**89**	86	82	88	80	88	86	86
SelfRegulationSCP1	79	84	82	82	84	88	82	**91**
UWaveGestureLibrary	88	88	75	89	91	**91**	91	88
SpokenArabicDigits	99	91	97	**99**	98	99	97	98
SelfRegulationSCP2	55	54	53	**59**	48	49	58	57
Japanese Vowels	97	99	93	96	72	92	97	98
Wins	2	3	2	3	3	4	4	**5**
Avg. Accuracy	78.1	72.8	74.4	78.5	72.5*	77.6	79.6	**80.9**
Total Correct	7467	7141	7118	7363	5414*	7569	7431	**7799**

Table 14: Performance comparison across datasets in %. Best results per dataset are boldfaced, and the best count is reflected in the win statistics. PenDigits and PhonemeSpectra are omitted from the finetuned comparisons, due to the size of these datasets, and the associated training compute and time required. *MOMENT does not report scores for CharacterTrajectories, and we exclude it while calculating MOMENT's scores.

K.2 Anomaly Detection

K.2.1 UCR Anomaly Detection Benchmark

Dataset The UCR anomaly detection dataset Dau et al. (2019) is a popular open-source univariate anomaly detection dataset. The dataset consists of >100 datasets from varying domains. We restrict ourselves to the same subset of 46 datasets used in MOMENT (Goswami et al., 2024) which cover a diverse set of sources.

Task Description Each dataset in the UCR archive is provided with a "clean" train split, and a corresponding test split. The standard setup in this task involves training a model to reconstruct clean samples (i.e. with no anomalies), and then use this model on the test set to reconstruct the data. The mean squared error is computed in a point-wise sense on all timestamps in the test set. If the error corresponding to each timestamp exceeds a certain threshold, we classify that timestamp as anomalous. For a fair comparison, we use the same sweep over the error thresholds as used in MOMENT, which uses 100 samples on a linearly spaced grid from the lowest error to the highest error

in the test set errors across all timestamps. Then, we compute an adjusted F1 score, which is standard practice in benchmarking anomaly detection models, for each threshold, and report the best adjusted F1 score for each dataset. For this experiment, the embedding model is frozen and only the linear reconstruction head is trained.

Downstream Setup Our model is extended with a reconstruction head, which consists of a linear layer that maps embeddings back to the raw time series values, i.e. $Z \in \mathbb{R}^{T' \times 1 \times H} \rightarrow Z_t \in \mathbb{R}^T$. We empirically observe better results by applying an `AvgPool` on the embeddings (with a stride of 8) before the reconstruction head, as it potentially reduces the high fidelity of our per time point embeddings. Consequently, the linear layer is then of dimensions $\mathbb{R}^{H \times 8}$.

Baselines To ensure a fair comparison with models from varying classes, i.e. task-specific vs general representation learning, we benchmark ourselves against the same set of models used in `MOMENT`, which consist of state-of-the-art anomaly detection models, as well as general representation learning methods. These consist of: `Anomaly Detection Transformer`, `DGHL`, `GPT4TS`, `TimesNet` and `MOMENT`. Given our limited compute availability, all baseline results reported in the results table are drawn from prior published work and we limited ourselves to reported models in the `MOMENT` paper. We exclude `T-Rep` and `TS2Vec`, as they do not report results on this dataset/task.

Metrics We evaluate performance by measuring an adjusted F1 score for each dataset after optimizing the threshold for each dataset separately.

Results We report adjusted F1 scores across all datasets and models in Table 15. While model performance varies considerably by dataset, CHARM achieves strong overall results in terms of both average F1 score and total wins.

Dataset	Anomaly Transformer	MOMENT	CHARM	DGHL	GPT4TS	TimesNet
1sddb40	0.03	0.54	0.99	0.39	0.19	0.68
BIDMC1	0.99	1.00	1.00	1.00	1.00	1.00
CHARISfive	0.01	0.13	1.00	0.02	0.02	0.08
CHARISten	0.02	0.11	0.12	0.01	0.01	0.03
CIMIS44AirTemperature3	0.06	0.98	0.98	0.50	0.18	0.47
CIMIS44AirTemperature5	0.39	0.99	0.85	0.96	0.20	0.71
ECG2	1.00	1.00	1.00	0.62	0.90	1.00
ECG3	0.36	0.98	0.93	0.80	0.84	0.48
Fantasia	0.75	0.95	0.97	0.66	0.87	0.55
GP711MarkerLFM5z4	0.93	1.00	0.64	0.50	0.64	0.95
GP711MarkerLFM5z5	0.76	0.97	0.75	0.31	0.48	0.90
InternalBleeding5	0.94	1.00	1.00	1.00	0.92	1.00
Italianpowerdemand	0.01	0.74	0.17	0.59	0.01	0.44
Lab2Cmac011215EPG5	0.99	0.98	1.00	0.34	0.60	0.99
Lab2Cmac011215EPG6	0.41	0.10	0.12	0.26	0.10	0.17
MesoplodonDensirostris	1.00	0.84	1.00	0.79	1.00	1.00
PowerDemand1	0.87	0.44	0.43	0.49	0.76	0.95
TkeepFirstMARS	0.02	0.15	0.03	0.02	0.02	0.23
TkeepSecondMARS	0.83	1.00	1.00	0.16	0.12	0.95
WalkingAceleration5	1.00	1.00	0.89	0.48	1.00	0.96
apneaecg	0.40	0.20	0.44	0.25	0.31	0.26
apneaecg2	0.65	1.00	0.92	1.00	1.00	0.90
gait1	0.18	0.36	0.53	0.51	0.48	0.47
gaitHunt1	0.08	0.43	0.99	0.02	0.10	0.30
insectEPG2	0.12	0.23	0.73	0.14	0.81	0.96
insectEPG4	0.98	1.00	0.70	0.46	0.21	0.85
lstdbs30791AS	1.00	1.00	1.00	1.00	1.00	1.00
mit14046longtermecg	0.45	0.59	0.98	0.43	0.97	0.97
park3m	0.15	0.64	0.61	0.20	0.63	0.93
qtdbSel1005V	0.41	0.65	0.75	0.44	0.39	0.90
qtdbSel100MLII	0.42	0.84	0.90	0.41	0.60	0.87
resperation1	0.16	0.15	0.83	0.03	0.59	0.96
s20101mML2	0.69	0.71	1.00	0.15	0.05	0.08
sddb49	0.89	1.00	1.00	0.88	0.94	1.00
sel840mECG1	0.41	0.66	1.00	0.32	0.28	0.36
sel840mECG2	0.15	0.39	0.60	0.32	0.28	0.21
tilt12744mtable	0.07	0.24	0.14	0.04	0.05	0.16
tilt12754table	0.23	0.64	0.04	0.04	0.06	0.14
tiltAPB2	0.92	0.98	1.00	0.36	0.83	0.38
tiltAPB3	0.17	0.85	0.62	0.03	0.05	0.29
weallwalk	0.00	0.58	1.00	0.07	0.13	0.17
Wins	5	18	**24**	4	5	12
Average	0.485	0.684	**0.754**	0.415	0.479	0.627

Table 15: Anomaly detection performance across 46 UCR datasets

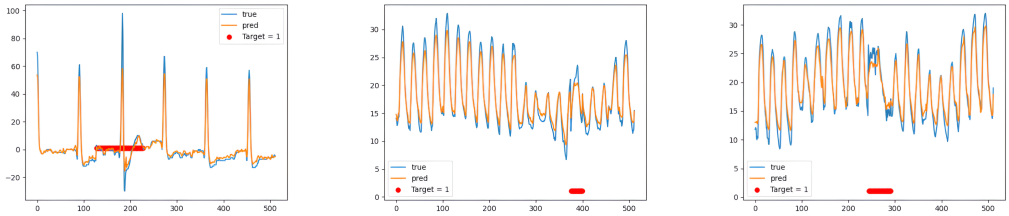


Figure 14: UCR Anomaly test set reconstructions visualized, with anomalous regions highlighted for `1sddb40`, `CIMIS44AirTemperature3` and `CIMIS44AirTemperature5`. `true` refers to the ground truth values, while `pred` refers to our reconstruction head's predictions.

K.2.2 Skoltech Anomaly Detection Benchmark (SKAB)

Dataset To evaluate our performance on a real world industrial setup, we use the open-source Skoltech Anomaly Benchmark suite Katser & Kozitsin (2020), which consists of a point-wise anomaly detection task using data from 8 sensors attached to a mechanical testbed. The dataset itself consists of 34 sub-dataset instances consisting of both outlier detection, and changepoint detection anomalies.

Task Description We follow the standard setup accompanying the SKAB benchmark for our model, as well as reproducing other baselines on this dataset. This involves splitting the data into several train and test sets, where each instance in the train and test set is trained with a fresh model to reconstruct the training dataset, i.e. minimize MSE on the reconstruction task $||x_{\text{train}} - \hat{x}_{\text{train}}||_2$, and then evaluated by computing the reconstruction of the corresponding test instance, $\hat{x}_{\text{test}}$. Based on the train set reconstruction, we compute the Upper Control Limit (UCL), based on the 99th percentile quantiles, and apply an adjustment factor of $\frac{4}{3}$. Then, for the reconstructed test data $\hat{x}_{\text{test}}$, we classify anomalies if the absolute values of the residuals, i.e. $||x_{\text{test}} - \hat{x}_{\text{test}}||$ lie outside the UCL limit. This exact anomaly detection setup is commonly applied to all baseline models in the test suite.

Downstream Setup Similar to K.2.1, we rely on training a linear head to reconstruct"clean" training data. I.e., we use a single linear layer $\mathbb{R}^{H \times 1}$ to map our embeddings Z back to the raw time series values. The hyperparameters used for training the linear head are listed in 16.

Hyperparameter	Value
Optimizer	AdamW
Weight Decay	None
Learning Rate	1e-3
Epochs	1000

Table 16: Hyperparameters to train reconstruction head for anomaly detection

Baselines To compare ourselves to a diverse set of models, we include all baselines available in SKAB leaderboard, as well as **`T-Rep`**, **`TS2Vec`**, **`MOMENT`**.

The baselines in the SKAB leaderboard come from a diverse set of modeling approaches, which rely on both statistical techniques, as well as more modern CNN/LSTM based methods. We list brief descriptions of the SKAB baselines here:

- **Hotelling's T-squared statistic**: Measures the Mahalanobis distance of new samples from the mean using variances for multivariate process monitoring.
- **Hotelling's T-squared + Q statistic (PCA-based)**: Uses principal component analysis, where T^2 captures variation in the principal subspace and Q measures residuals, combined via logical OR for monitoring.

- **Isolation Forest (iForest)**: An ensemble-based method that isolates anomalies as points with short average path lengths in random trees.
- **LSTM-based Neural Network**: An LSTM network trained for anomaly detection using reconstruction error as the anomaly score.
- **Feed-Forward Autoencoder**: A standard autoencoder that detects anomalies via reconstruction error in vector data.
- **Convolutional Autoencoder (Conv-AE)**: A CNN-based autoencoder for anomaly detection in time series via reconstruction error.
- **LSTM Autoencoder (LSTM-AE)**: A sequence-to-sequence LSTM autoencoder that reconstructs temporal patterns and flags anomalies via reconstruction error.
- **LSTM Variational Autoencoder (LSTM-VAE)**: A probabilistic LSTM autoencoder that models latent distributions and detects anomalies using reconstruction error.
- **Variational Autoencoder (VAE)**: A generative model that learns latent variable distributions of input data, with anomalies identified via reconstruction error.
- **MSCRED**: A multi-scale convolutional recurrent encoder-decoder that reconstructs signature matrices of system statuses and uses residuals to detect anomalies.
- **MSET**: A nonparametric statistical modeling technique that estimates values via weighted averages of historical data for anomaly detection.

Reproducing Baselines To ensure a fair comparison, we reproduce the baseline methods `T-Rep`, `TS2Vec`, and `MOMENT` following the protocols described below:

1. **`T-Rep`:** We train the model using the self-supervised contrastive loss used in the paper, on each dataset instance using the official implementation and author-recommended hyperparameters. Subsequently, we append a linear reconstruction head, which is trained using the hyperparameters specified in Table 16. The base encoder remains frozen during this stage.
2. **`TS2Vec`:** We adopt an identical procedure to that of `T-Rep`, i.e., training with the official implementation and hyperparameters, followed by the addition of a frozen base encoder with a trainable linear reconstruction head.
3. **$\texttt{MOMENT}_0$:** We directly evaluate the `AutonLab/MOMENT-1-large` checkpoint in *reconstruction mode*. This configuration utilizes the reconstruction head employed during pretraining and is applied to the test set without any further training or fine-tuning.
4. **$\texttt{MOMENT}_{\texttt{LP}}$:** We employ the same checkpoint in *embedding mode*, in which representations are extracted and paired with a linear reconstruction head. The linear head is trained on the training instances using the hyperparameters from Table 16, consistent with the setup applied to `CHARM`, `T-Rep`, and `TS2Vec`.

Note that the `MOMENT` model was not pre-trained on the SKAB dataset, whereas our model was. This suggests that there may be additional untapped performance potential for `MOMENT` on this benchmark, since pre-training it on SKAB could plausibly improve its results. However, due to the large computational demands of `MOMENT` and our limited access to compute resources, we were unable to conduct this experiment.

Metrics We reported the average F1 score over all instances, as well as the `False Alarm Rate` (FAR) and `Missed Alarm Rate` (MAR), for all baseline models. We outline the mathematical representation of these terms, and their relation to commonly used binary classification metrics here:

$$\text{Missed Alarm Rate (FNR)} = \frac{FN}{TP + FN} = 1 - \text{Recall}$$

$$\text{Specificity (TNR)} = \frac{TN}{TN + FP}$$

$$\text{False Alarm Rate (FPR)} = \frac{FP}{TN + FP} = 1 - \text{Specificity}$$

Results We compile our results on the SKAB benchmark along with the different baseline models collected in separate classes (Self-Supervised vs Classical) in Table 17. Similar to UCR, we here also observe strong performance for CHARM.

Table 17: Comparison of anomaly detection performance across baselines and our method. Higher F1 scores are better (↑), while lower False Alarm Rate (FAR) and Missed Alarm Rate (MAR) are better (↓).

Category	Method	F1 ↑	FAR (%) ↓	MAR (%) ↓
Representation Learning	T-Rep	0.78	12.60	28.51
	MOMENT_0	0.79	14.20	26.98
	TS2Vec	0.79	12.77	27.61
	$\text{MOMENT}_{\text{LP}}$	0.82	15.52	20.73
Classical	Conv-AE	0.78	13.55	28.02
	MSET	0.78	39.73	14.13
	T-squared+Q (PCA)	0.76	26.62	24.92
	Isolation Forest	0.29	**2.56**	82.89
	LSTM-VAE	0.56	9.13	55.03
	MSCRED	0.36	49.94	69.88
	CHARM	**0.86**	19.35	**12.69**

K.3 Forecasting

Datasets We evaluated our model on the benchmarks introduced in Autoformer Wu et al. (2021), which have become standard multivariate time series forecasting benchmarks. Specifically, the benchmark suite includes the Electricity Transformer Dataset (ETT), Weather, Exchange Rate and Illness datasets. The model pretraining included the train splits of these datasets (Section I).

The train/valid/test split is identical to the standard protocol in the other baselines we compare with, which is a 6/2/2 split for the ETT datasets, and a 7/1/2 split for all other datasets.

To ensure a fair comparison, we adopt the standard set of lookback horizons and future horizon values across all forecasting datasets, as specified in Table 19. While earlier works primarily use a lookback horizon of 96, more recent studies have also incorporated a longer lookback horizon of 512. To maintain consistency and comparability, we therefore report our linear probing results under both lookback settings. Furthermore, since different papers also employ different prediction horizons, we follow each work's choice of horizons to respect their experimental setup and allow for direct comparison.

Task Description Forecasting tasks consist of taking a window of time series data and predicting future time steps. Formally, given an input of dimensions $(T_h \times C)$, where T_h denotes the "lookback" horizon, the goal is to predict the future T_f time steps for all channels.

Downstream Setup We use the embeddings of the input horizon data, stack the embeddings across all time steps for each channel, and train the model to minimize an aggregate loss metric[6] between the predicted and true values for each channel.

Since the pretraining task was not designed for direct linear forecasting, nor to produce single-step predictions, we evaluate forecasting performance using the following modeling approaches:

1. **CHARM+LP** A per-dataset, per-channel, per-horizon linear regression head is trained on top of frozen embeddings.
2. **CHARM+NLH:** A common non-linear prediction head is trained across all datasets, channels, and horizons, with the encoder kept frozen.
3. **CHARM+NLH FT:** The full model (encoder + non-linear prediction head) is trained end-to-end, shared across datasets, channels, and horizons.

[6] $\textbf{loss} = \frac{\textbf{MSE}+\textbf{MAE}}{2}$

Non-Linear Head (NLH) The head is designed to first mix information across both time and channels, then refine within each channel, and finally project to the forecasting horizon:

- Transformer across time & channels ($n_{\text{heads}} = 4$, $n_{\text{layers}} = 2$, hidden dimension 2048).
- Transformer per channel ($n_{\text{heads}} = 4$, $n_{\text{layers}} = 1$, hidden dimension 2048).
- Per-channel linear projection to a maximum horizon of 720.

It is important to note that in the non-linear head setup, the forecasting module is shared across all datasets, channels, and horizons. The transformer and projection layers are not customized or tuned for any specific dataset, horizon, or channel, ensuring a single common forecasting head is used throughout. If the target horizon T_f is less than the max horizon (720), we simply apply the loss to the first T_f predictions from the head.

Training protocol. All non-linear head models were trained on the full CHARM dataset collection (Section I) without hyperparameter optimization due to resource constraints. The training setup is summarized in Table 18. The linear heads were trained for each (dataset, horizon, channel) combination separately, which is standard for a linear probing setup in time series forecasting, and in line with other baseline implementations. To this end, we conducted hyperparameter optimization as reported in table 20 and present the best results.

Hyperparameter	Value
Lookback horizon	512
Datasets	All CHARM datasets (Section I)
Batch size	256 (gradients accumulated across datasets)
Epoch definition	10 steps across 4 nodes
Max epochs	60 (early stopping, patience = 5)
Optimizer	AdamW
Loss	$(\text{MSE} + \text{MAE})/2$
Schedule	Cosine
Learning rate	1×10^{-3}
Weight decay	0.01

Table 18: **Training protocol for non-linear forecasting (NLH) heads.**

Dataset	Lookback Horizon T_h	Target Horizon T_f
`ETTh1`	96/512	{24, 48, 168, 336, 720}
`ETTh2`	96/512	{24, 48, 168, 336, 720}
`ETTm1`	96/512	{24, 48, 96, 288, 672}†
`ETTm2`	96/512	{24, 48, 96, 288, 672}†
Weather	96/512	{96, 192, 336, 672}
Exchange Rate	96/512	{96, 192, 336, 672}
Illness	96/512	{24, 36, 48, 60}

Table 19: Forecasting task specifications. †Some papers adopt the same prediction horizons as `ETTh1/2` for `ETTm1/2`.

Hyperparameter	Value
Optimizer	AdamW
Weight Decay	[1e-2, 1e-4]
Learning Rate	[1e-2, 1e-4]
Epochs	1000
LR Schedule	ReduceLROnPlateau
Reduction Factor	0.1
Early Stopping : Patience	50
Early Stopping : Tolerance	1e-6

Table 20: **Hyper-parameters to train linear prediction heads for forecasting tasks**

Baselines To ensure a fair assessment, we distinguish between three categories of methods: representation learning methods, reconstruction-based methods, and hybrid approaches.

- Representation learning methods focus on extracting meaningful embeddings of the data, independent of the reconstruction objective.
- Reconstruction-based methods emphasize the model's ability to directly predict or reconstruct future values.
- Hybrid approaches combine both ideas: they primarily rely on reconstruction-based training but additionally evaluate the representational power of the learned embeddings.

An important distinction is in the evaluation protocol. Both representation learning methods and hybrid approaches employ linear probing to assess the forecasting power of the embeddings. In contrast, reconstruction-based methods directly evaluate the pretrained model, since their pretraining task is already aligned with forecasting.

To establish a strong baseline, we compare against SOTA foundational time series models from each category. For pure representation learning methods, we include T-REP Fraikin et al. (2024) and TS2Vec Yue et al. (2022). For hybrid methods, we consider `MOMENT` Goswami et al. (2024) and `PatchTST` Nie et al. (2023b). Finally, for reconstruction-based methods, we evaluate `Toto` Cohen et al. (2025), `TIMEMIXER++` Wang et al. (2025), `Moirai` Woo et al. (2024a), and `VisionTS` Chen et al. (2025). We exclude results from works such as `TimesFM` Das et al. (2024), `UniTS` Gao et al. (2024b) and `Chronos` Ansari et al. (2024), as their experimental setups differ substantially from ours, making direct comparison infeasible.

Reproducing Baselines We reproduce the baseline methods `T-Rep` and `TS2Vec` on the `Weather`, `ILI`, and `Exchange Rate` datasets following the original papers' pretraining setup. Specifically, each model is first pretrained on the respective dataset, after which a linear forecasting head is added and trained while keeping the base model frozen. The forecasting head is trained using the same architecture and hyperparameters as specified in the original paper's downstream forecasting setup. For the `ETT` datasets, results for both models are taken directly from the original `T-Rep` paper (Fraikin et al., 2024). For reconstruction-based and hybrid models, we report the scores as presented in their respective papers for the corresponding datasets and horizons. The compiled results are shown in Table 22.

Metrics We quantitatively assess the model's performance using mean squared error (MSE) and mean absolute error (MAE) metrics averaged over all forecasted time steps and across all target variables, which is standard practice for multivariate forecasting benchmarks.

Results The results comparing CHARM with other state-of-the-art representation learning methods, along with the reproduced baselines, are summarized in Table 21. These results underscore the strong performance of CHARM embeddings relative to competing methods in this category. Further, Table 22 demonstrates that CHARM remains competitive with hybrid and reconstruction-based models—including substantially larger models trained on significantly larger datasets (e.g., TOTO, Moirai).

Dataset	H	T-Rep		TS2Vec		CHARM+LP	
		MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.511	0.496	0.575	0.529	**0.310**	**0.350**
	48	0.546	0.524	0.608	0.553	**0.358**	**0.376**
	168	0.759	0.649	0.782	0.659	**0.451**	**0.430**
	336	0.936	0.742	0.956	0.753	**0.517**	**0.466**
	720	1.061	0.813	1.092	0.831	**0.546**	**0.498**
ETTh2	24	0.560	0.565	0.448	0.506	**0.186**	**0.267**
	48	0.847	0.711	0.685	0.642	**0.242**	**0.303**
	168	2.327	1.206	2.227	1.164	**0.391**	**0.396**
	336	2.665	1.324	2.803	1.360	**0.430**	**0.427**
	720	2.690	1.365	2.849	1.436	**0.470**	**0.466**
ETTm1	24	0.417	0.420	0.438	0.435	**0.218**	**0.283**
	48	0.526	0.484	0.582	0.553	**0.282**	**0.324**
	96	0.573	0.516	0.602	0.537	**0.316**	**0.347**
	288	0.648	0.577	0.709	0.610	**0.395**	**0.391**
	672	0.758	0.649	0.826	0.687	**0.482**	**0.441**
ETTm2	24	0.172	0.293	0.189	0.310	**0.099**	**0.192**
	48	0.263	0.377	0.256	0.369	**0.131**	**0.223**
	96	0.397	0.470	0.402	0.471	**0.172**	**0.253**
	288	0.897	0.733	0.879	0.724	**0.284**	**0.326**
	672	2.185	1.144	2.193	1.159	**0.403**	**0.400**
Weather	96	0.195	0.280	1.672	0.904	**0.158**	**0.199**
	192	0.235	0.316	1.569	0.894	**0.207**	**0.246**
	336	0.288	0.359	2.075	1.064	**0.265**	**0.287**
	672	0.362	0.402	2.828	1.305	**0.347**	**0.340**
Exchange Rate	96	1.180	0.806	0.462	0.544	**0.084**	**0.203**
	192	3.947	1.344	0.968	0.765	**0.182**	**0.302**
	336	6.683	1.699	1.759	1.037	**0.353**	**0.429**
	720	3.900	1.504	2.266	1.184	**0.929**	**0.727**
ILI	24	3.631	1.227	3.463	1.173	**2.799**	**1.080**
	36	3.979	1.313	3.889	1.282	**1.754**	**0.797**
	48	4.290	1.363	4.219	1.339	**1.699**	**0.820**
	60	4.361	1.375	4.198	1.329	**1.740**	**0.838**

Table 21: Representation learning only Long-horizon forecasting results across datasets. Input length = 96. Lower is better. **Bold** = best, Underline = second best. We use a frozen encoder with a linear head for this experiment.

Dataset	H	Rec.												Hybrid				Ours					
		Toto		Moirai_S		Moirai_B		Moirai_L		TimeMixer++		VisionTS		MOMENT-LP		PatchTST-LP		CHARM+LP		CHARM + NLH		CHARM + NLH FT	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	96	0.382	**0.381**	0.375	0.402	0.384	0.402	0.380	0.398	0.361	0.403	**0.353**	0.383	0.387	0.410	0.371	0.400	0.452	0.464	0.467	0.478	0.465	0.464
	192	0.428	0.408	0.399	0.419	0.425	0.429	0.440	0.434	**0.375**	**0.400**	0.392	0.410	0.410	0.426	0.411	0.428	0.502	0.503	0.536	0.523	0.512	0.498
	336	0.457	**0.422**	0.422	0.429	0.450	0.456	0.514	0.474	0.416	0.441	**0.407**	0.423	0.422	0.437	0.445	0.446	0.565	0.540	0.600	0.564	0.560	0.530
	720	0.472	0.440	**0.413**	0.444	0.470	0.473	0.705	0.568	0.430	0.434	0.416	**0.405**	0.454	0.416	0.487	0.478	0.699	0.622	0.764	0.657	0.693	0.612
ETTh2	96	0.273	**0.310**	0.281	0.334	0.277	0.327	0.287	0.325	0.276	0.328	0.271	0.328	0.288	0.345	0.285	0.344	0.243	0.334	**0.236**	0.328	0.240	0.330
	192	0.339	**0.356**	0.340	0.373	0.340	0.374	0.347	0.367	0.342	0.379	0.328	0.367	0.349	0.386	0.356	0.387	**0.294**	0.368	0.296	0.369	0.298	0.369
	336	0.410	0.387	0.362	0.393	0.371	0.401	0.377	0.393	0.346	0.398	0.345	0.381	0.369	**0.377**	0.425	0.377	0.334	0.394	0.340	0.398	**0.332**	0.391
	720	**0.375**	**0.400**	0.380	0.416	0.394	0.426	0.404	0.421	0.392	0.415	0.388	0.422	0.403	0.439	0.395	0.434	0.424	0.448	0.421	0.446	0.395	0.432
ETTm1	96	0.320	**0.333**	0.404	0.383	0.335	0.360	0.353	0.363	0.310	0.334	0.341	0.347	0.293	0.349	**0.292**	0.348	0.337	0.386	0.341	0.387	0.337	0.382
	192	0.371	0.364	0.435	0.402	0.379	0.402	0.376	0.380	0.348	0.362	0.360	**0.360**	**0.326**	0.368	0.329	0.369	0.392	0.419	0.398	0.423	0.390	0.412
	336	0.408	0.388	0.462	0.416	0.394	0.416	0.399	0.395	0.376	0.391	0.377	0.374	**0.352**	0.384	0.364	0.391	0.434	0.442	0.437	0.449	0.434	0.440
	720	0.485	0.426	0.490	0.437	0.419	0.437	0.432	0.417	0.440	0.423	0.416	**0.405**	**0.405**	0.416	0.415	0.419	0.491	0.482	0.489	0.488	0.484	0.478
ETTm2	96	0.172	**0.237**	0.205	0.282	0.195	0.269	0.189	0.260	0.170	0.245	0.228	0.282	0.170	0.260	0.167	0.257	0.154	0.255	0.155	0.255	**0.150**	0.254
	192	0.232	0.280	0.318	**0.261**	0.303	0.300	0.247	0.300	0.229	0.291	0.262	0.305	0.227	0.297	0.229	0.300	0.188	0.282	0.197	0.287	**0.186**	0.283
	336	0.290	0.320	0.355	0.319	0.333	0.334	0.334	0.334	0.303	0.343	0.293	0.328	0.275	0.328	0.289	0.343	0.223	**0.309**	0.236	0.317	**0.220**	**0.309**
	720	0.372	0.375	0.410	0.415	0.377	0.372	0.372	0.386	0.373	0.399	0.343	0.370	0.363	0.387	0.363	0.386	**0.271**	**0.346**	0.294	0.356	0.278	0.350
Weather	96	0.149	0.179	0.173	0.212	0.167	0.203	0.177	0.208	0.155	0.205	0.220	0.257	0.154	0.209	0.158	0.209	0.151	0.198	0.150	0.196	**0.147**	**0.190**
	192	0.192	0.223	0.216	0.250	0.209	0.241	0.219	0.249	0.201	0.245	0.244	0.275	0.197	0.248	0.203	0.249	0.197	0.240	0.198	0.239	**0.191**	**0.232**
	336	0.245	**0.265**	0.260	0.282	0.256	0.276	0.292	0.277	**0.237**	**0.265**	0.280	0.299	0.246	0.285	0.251	0.285	0.250	0.279	0.249	0.279	0.240	0.272
	720	**0.310**	**0.312**	0.320	0.322	0.321	0.323	0.365	0.350	0.312	0.334	0.330	0.337	0.315	0.322	0.321	0.336	0.324	0.332	0.324	0.334	**0.310**	0.324

Table 22: Long-horizon forecasting results across datasets. Input length = 512. Lower is better. **Bold** = best, Underline = second best. Last three columns are our CHARM variants; *CHARM+LP* is a **Rep.** approach, while *CHARM + NLH* and *CHARM + NLH FT* are **Rec.** approaches.

Visualizations We present a sample of forecasting results from **CHARM+LP** and using a lookback window of 96.

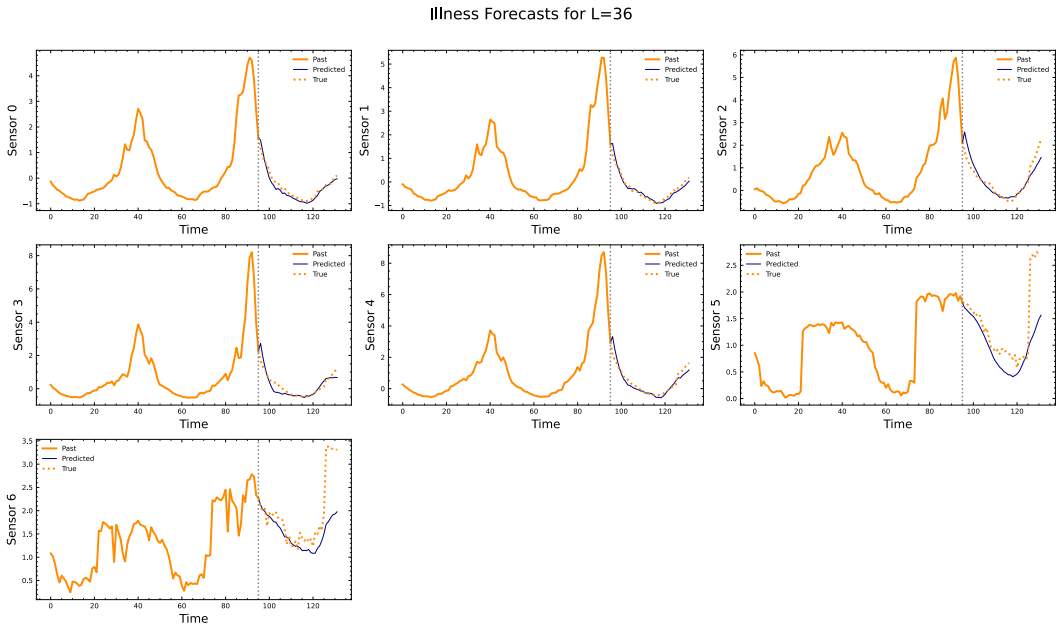


Figure 15: Illness Forecasts

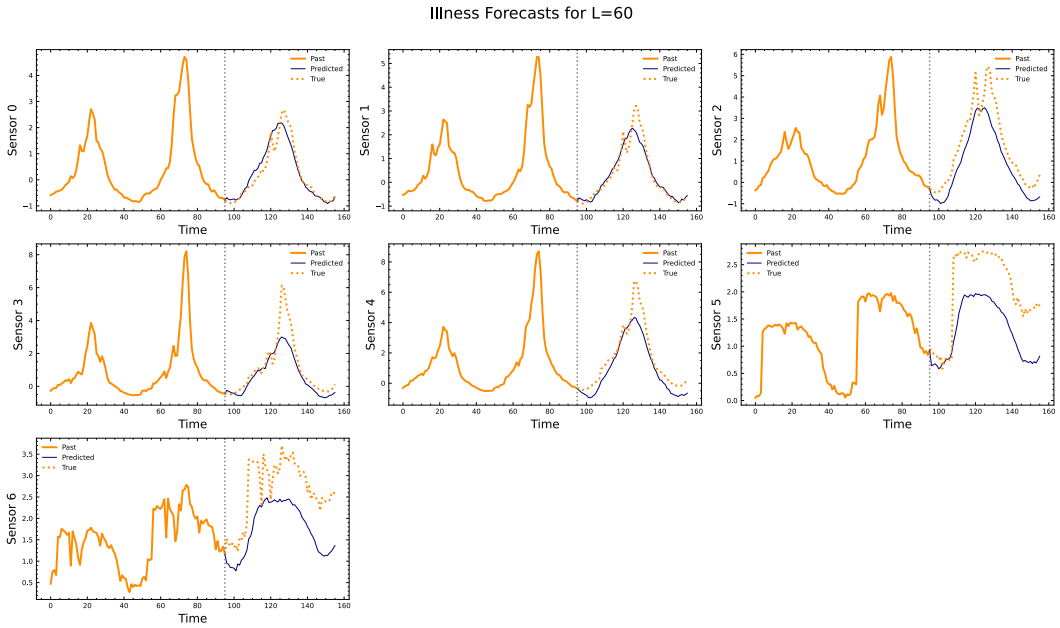


Figure 16: Illness Forecasts

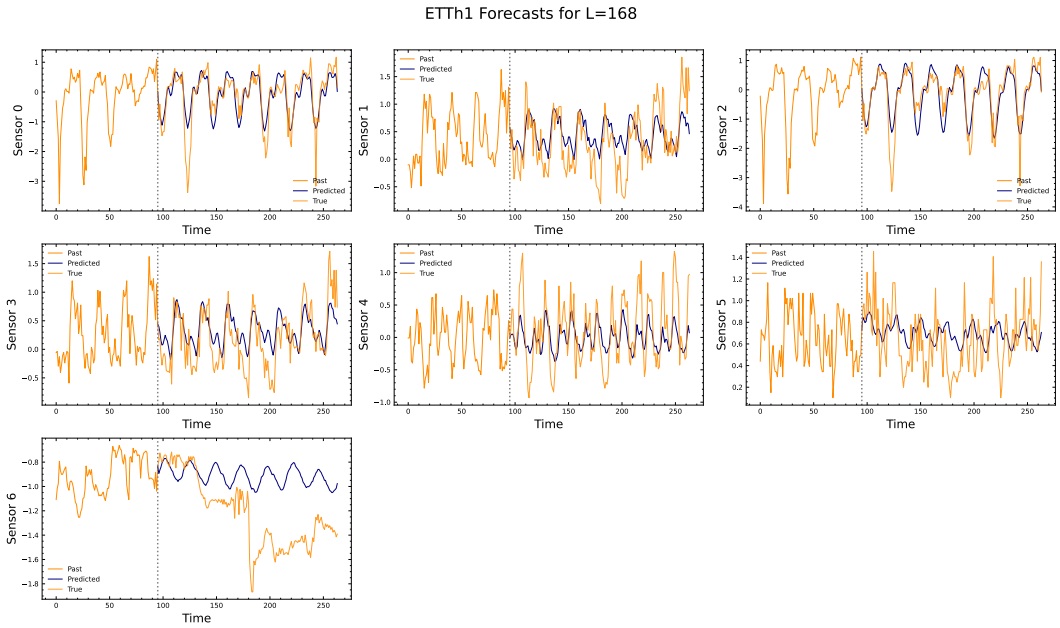


Figure 17: ETTh1 Forecasts

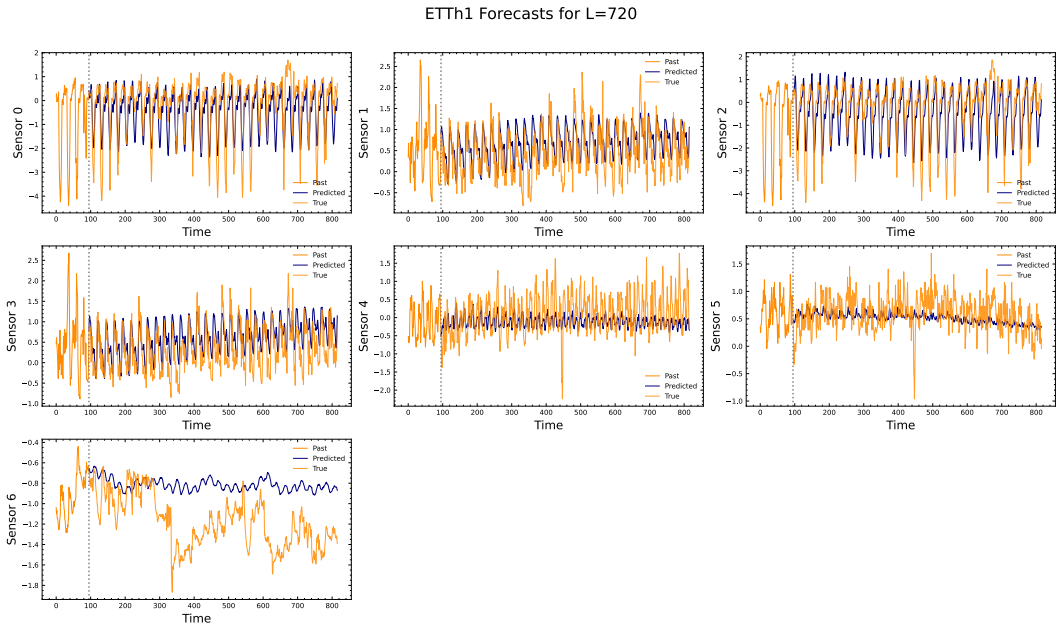


Figure 18: ETTh1 Forecasts

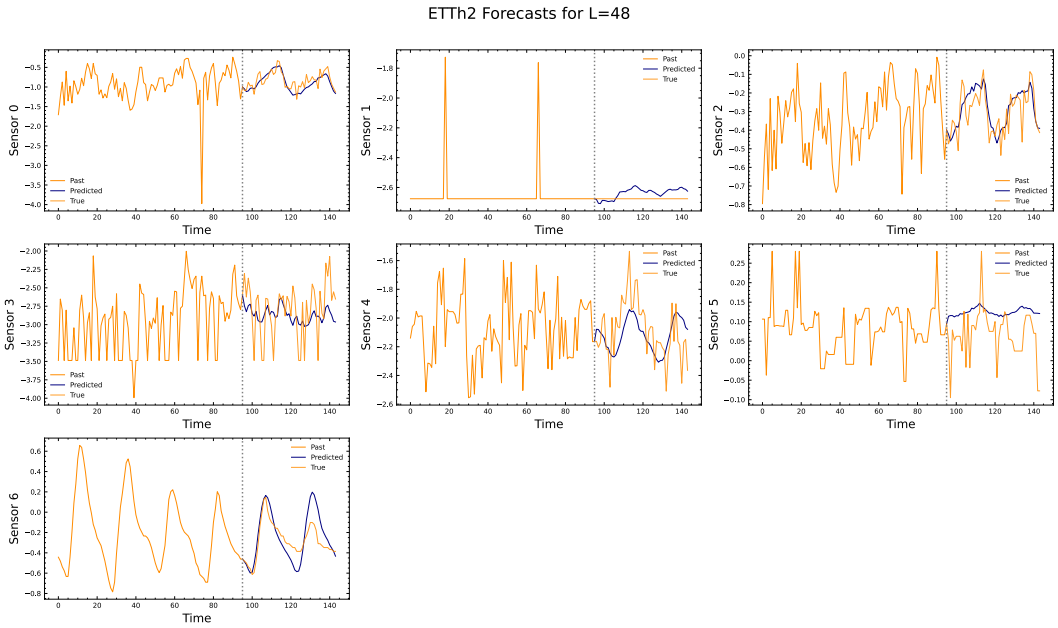


Figure 19: ETTh2 Forecasts

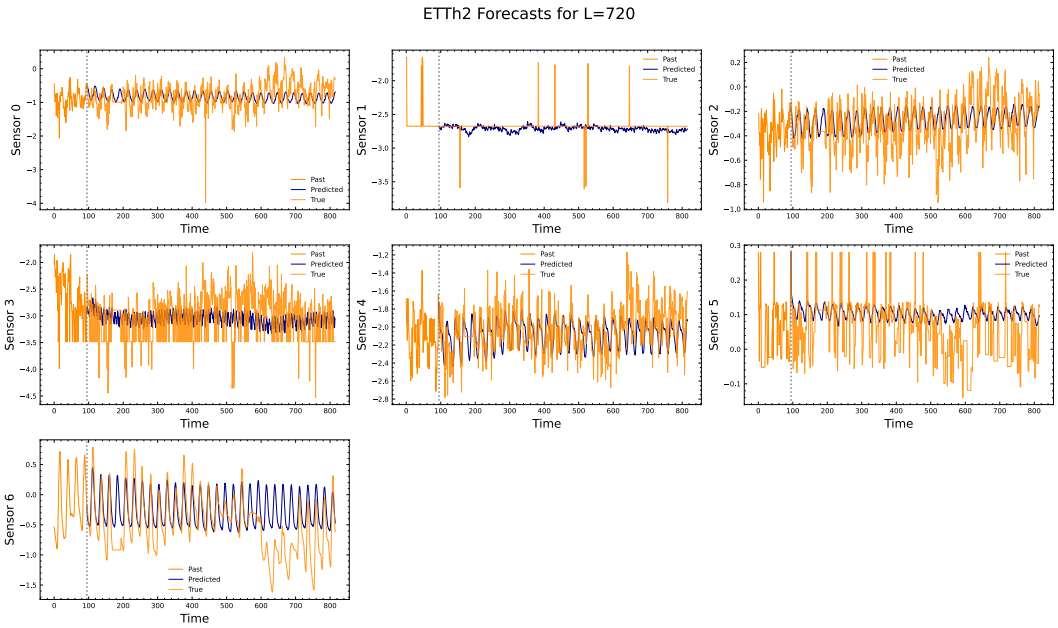


Figure 20: ETTh2 Forecasts

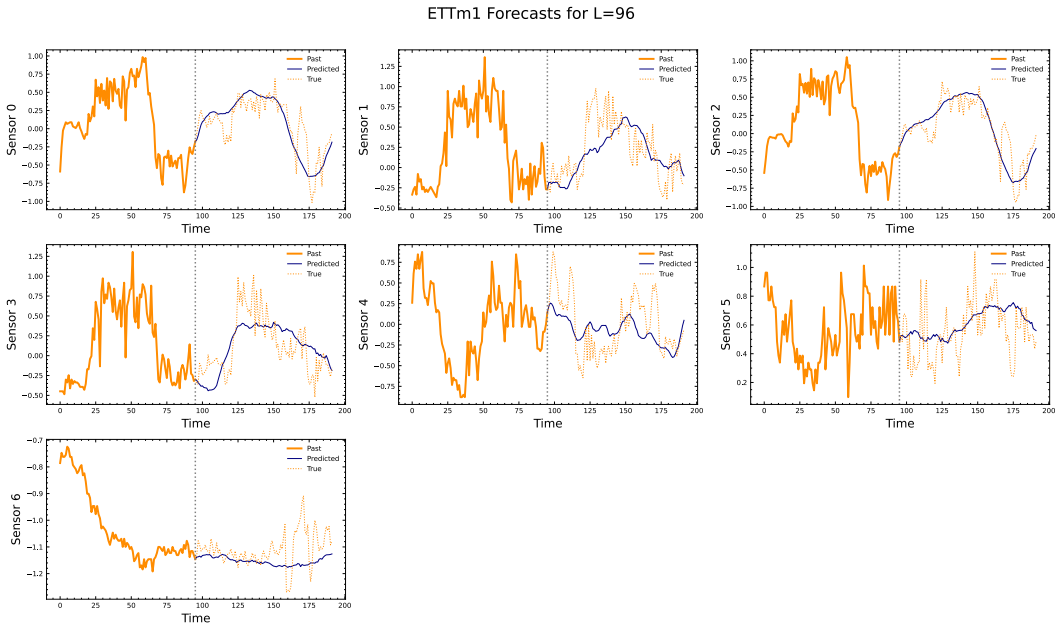


Figure 21: ETTm1 Forecasts

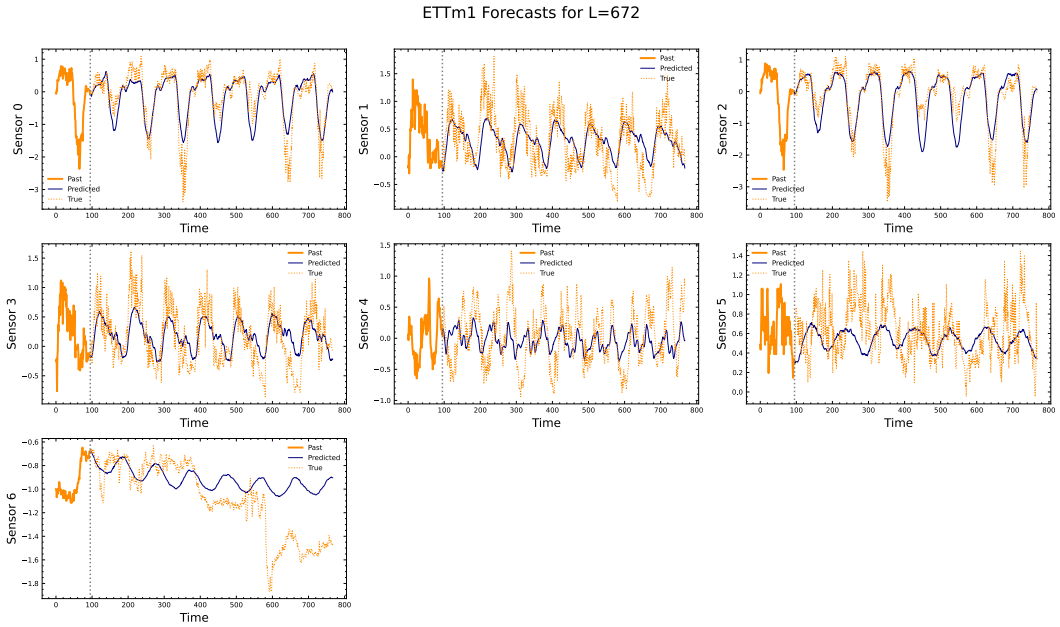


Figure 22: ETTm1 Forecasts

Linear Probing and Pooling Ablations Experiments related to pooling strategies, shown in Table 23, were designed to contrast different pooling approaches commonly used to aggregate data in other domains and to examine their applicability to time series. We also experimented with an MLP to investigate whether the embeddings benefit more from a non-linear model, since the training objective is not aimed at maximizing linear predictability. We only compare these methods using the ETTh1/2 datasets with a lookback window of 96 and forecasting horizons of $24, 48, 168, 336, 720$.

To further study the effect of different pooling strategies, we evaluated the following three approaches:

1. **Flattening**: The embeddings are flattened into a single vector representation for each channel.
$$Z : \mathbb{R}^{T \times H} \to Z_{\text{flat}} \in \mathbb{R}^{TH}$$
2. **Mean Pooling**: The embeddings are averaged over the time dimension (but not across channels), yielding an H-dimensional representation per channel.
$$Z : \mathbb{R}^{T \times H} \to Z_{\text{mean}} \in \mathbb{R}^{H}$$
3. **Last Time Step**: The embedding from the last time step of each channel is taken as the representative embedding.
$$Z : \mathbb{R}^{T \times H} \to Z_{-1} \in \mathbb{R}^{H}$$

In addition, we experimented with:

- **Frozen Encoder + 2-Layer MLP**: A per-dataset, per-channel, per-horizon MLP head (two linear layers with ReLU activations) trained on top of frozen embeddings.

Dataset	Pool	Head	24		48		168		336		720	
			MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	**none**^{1a}	linear^{1b}	**0.31**	**0.35**	0.36	**0.38**	0.45	**0.43**	0.52	0.47	0.55	**0.50**
		MLP^{2b}	0.32	0.36	**0.37**	0.38	**0.46**	0.44	**0.50**	**0.46**	**0.54**	**0.50**
	last time step^{2a}	linear^{1b}	0.39	0.39	0.43	0.41	0.51	0.46	0.53	0.47	0.54	0.50
		MLP^{2b}	**0.37**	0.38	0.41	0.40	0.49	0.45	0.54	0.48	0.55	0.51
	mean^{3a}	linear^{1b}	0.66	0.49	0.68	0.50	0.72	0.53	0.69	0.54	0.69	0.57
		MLP^{2b}	0.61	0.50	0.74	0.51	0.77	0.54	0.74	0.55	0.73	0.58
ETTh2	**none**^{1a}	linear^{1b}	**0.19**	**0.27**	**0.24**	**0.30**	**0.39**	**0.40**	**0.43**	**0.43**	**0.47**	**0.47**
		MLP^{2b}	0.20	**0.27**	0.25	0.31	0.40	**0.40**	0.46	0.44	0.51	0.48
	last time step^{1a}	linear^{1b}	**0.19**	0.28	0.25	0.31	0.40	**0.40**	0.44	**0.43**	0.49	0.48
		MLP^{2b}	0.20	0.28	0.26	0.32	0.40	**0.40**	0.45	0.44	0.49	0.48
	mean^{1a}	linear^{1b}	0.25	0.32	0.29	0.35	0.44	0.43	0.44	0.45	0.50	0.49
		MLP^{2b}	0.25	0.33	0.30	0.35	0.44	0.43	0.46	0.45	0.50	0.49

Table 23: Ablation results comparing pooling strategies and heads across ETTh1 and ETTh2. **Bolded** values denote best within each dataset and horizon. The encoder is kept frozen for this experiment.

From Table 23, we observe that no pooling (1a) combined with either a linear or MLP probe yields the best results on both `ETTh1` and `ETTh2`. Interestingly, we observe that for `ETTh1` using just the last time step's embedding (1b) yields competitive scores with an average increase of 8.7% (MSE) and 4.2% (MAE) when compared to no pooling (1a). Comparatively, mean pooling (1c) has an increase of 60.5% (MSE) and 24.4% (MAE) Similarly, for `ETTh2`, we observe that using the last time step embeddings (1b) has only a 0.85% (MSE) and 1.33% (MAE) increase in error, when compared to mean pooling which has a 9.32% (MSE) and 8.49% (MAE) increase in error.

This observation is in line with (Bardes et al., 2024), which demonstrated that using attentive probing to pool embeddings was empirically superior for downstream task performance compared to directly mean pooling the embeddings, which can potentially result in lossy, diffuse representations which fail to capture finer granularities in the data.

L Ablations

To better understand the effects of different components in our model, we perform a series of ablations involving the proposed architectural additions - (i) TCN featurization layer, (ii) text based attention mechanisms, (iii) effect of description quality, (iv) effect of embedding model, and (v) alternative approaches to multi modal text + time series. To quantify the effect of each of these changes, we measure performance on forecasting and classification by measuring the following quantities:

1. Total number of correct classifications for UEA

Category	Configuration	# Correct	Accuracy	ETTh1$_{T_f=168}$ MSE	ETTh1$_{T_f=168}$ MAE	ETTh2$_{T_f=168}$ MSE	ETTh2$_{T_f=168}$ MAE
Featurization Layer	w/ Patching	4200	63.0%	0.54	**0.58**	0.79	1.18
	TCN$_{\texttt{no text}}$	**4713**	**68.0%**	0.54	0.61	**0.62**	1.18
TCN Variants	TCN$_{\texttt{no text}}$	4713	68.0%	0.54	0.61	0.62	1.19
	TCN$_{\texttt{gate}}$	4732	69.0%	0.52	0.56	0.74	1.04
	TCN$_{\texttt{conv}}$	**4897**	**71.4%**	**0.42**	**0.49**	**0.57**	**0.80**
Text Attention	$\varnothing$ (self-attn)	4563	67.9%	0.46	0.50	0.64	0.93
	$\mathbf{\Delta}$	4643	68.6%	0.50	0.55	0.68	0.97
	$\mathbf{G}$	4785	69.8%	0.46	0.51	0.59	0.94
	$\mathbf{\Delta} + \mathbf{G}$	**4897**	**71.4%**	**0.42**	**0.49**	**0.57**	**0.80**

Table 24: **Unified ablation results across classification and forecasting tasks.** Top: comparison of TCN-based featurization against patch embeddings. Middle: architectural variants of the TCN module. Bottom: ablation of text-aware attention mechanisms. Forecasting performance is reported on `ETTh1` and `ETTh2` with horizon $T_f = 168$. Bold denotes the best configuration within each block. **Configuration details:** w/ Patching uses a vanilla patch embedding; TCN$_{\texttt{no text}}$ removes all text-conditioning layers. TCN$_{\texttt{gate}}$ applies text-based gating; TCN$_{\texttt{conv}}$ uses text-conditioned convolutions. $\mathbf{G}$ denotes text gating in the attention layer; $\mathbf{\Delta}$ denotes time-delta–aware attention. Unless otherwise specified, Featurization and TCN ablations use $\mathbf{\Delta} + \mathbf{G}$ attention, and Text Attention ablations use TCN$_{\texttt{conv}}$.

2. Average accuracy for UEA
3. Mean Squared Error for ETTh1 ($T_f = 168$)
4. Mean Absolute Error for ETTh1 ($T_f = 168$)
5. Mean Squared Error for ETTh2 ($T_f = 168$)
6. Mean Absolute Error for ETTh2 ($T_f = 168$)

For the ablation study, we train and probe the model with the following protocols:

Pretraining The base model is pretrained with a subset of datasets (UEA, ETTh, ETTm, Weather, Illness), for 50 epochs, with a learning rate of 5e-4.

Classification Evaluations The hyperparameters used for measuring classification performance are listed in Table 25. We use mean-pooled embeddings, i.e. $\bar{Z} \in \mathbb{R}^{B \times H}$, instead of flattened embeddings, $Z \in \mathbb{R}^{B \times T \times C \times H}$.

Forecasting Evaluations The forecasting setup is identical to the frozen linear model setup used in the downstream forecasting task. The hyperparameters are listed in Table 20. The embeddings are flattened for all timesteps and passed to a linear layer.

The ablation results for (i) and (ii) can be found in the main text in Table 24.

We cover the setup for the remaining ablations (iii), (iv), and (v) here, along with their results.

- **(iii) Effect of description quality**
  As channel descriptions are a first class citizen in training `CHARM`, we perform an ablation to investigate the effect of description quality on the model's performance. To this end we consider three cases:
  1) **Annotated descriptions**: manually curated sensor descriptions obtained from the official dataset metadata. These are obtained through either manual human annotation obtained by parsing the accompanying dataset metadata files, or are natively provided by the dataset provider.
  2) **Noisy descriptions**: high quality annotated descriptions, but with words dropped at random (with $p = 0.2$) during both training and evaluation.
  3) **Ordinal descriptions**: replace the annotated descriptions with structured, placeholder descriptions: [`Sensor1`, `Sensor2`, `Sensor3`...] for all datasets.

Hyperparameter	Value
C	{0.0001, 0.1, 1000}
kernel	{'rbf'}
degree	{3}
gamma	{'scale'}
coef0	{0}
shrinking	{True}
probability	{False}
tol	{0.001}
cache_size	{200}
class_weight	{None}
verbose	{False}
max_iter	{10000000}
decision_function_shape	{'ovr'}
random_state	{None}

Table 25: SVM Hyperparameter Grid for Ablations

- **(iv) Effect of text embedding model**
  We investigate the usage of different embedding models to assess the effect on downstream performance. We use 1) `nomic` (Nussbaum et al., 2025), 2) `minilm` (Wang et al., 2020), and 3) `mpnet` (Song et al., 2020) as representative models to assess the downstream impact on scores.
- **(v) Alternative multi-modal approaches**
  To investigate how naive multimodal approaches compare to our setup, we remove the text based layers altogether, and simply add in the channel description embeddings (from an LLM embedding model) in a pointwise sense to the time series embeddings. This is analogous to adding in position embeddings in the first layer of a vanilla transformer (Vaswani et al., 2017). We investigate adding these solely in the first layer, as well as in all layers. For numerical stability, we apply a `LayerNorm` on these embeddings to ensure they are of an appropriate scale.

Configuration	# Correct	Accuracy	$\texttt{ETTh1}_{168}$ MSE	$\texttt{ETTh1}_{168}$ MAE	$\texttt{ETTh2}_{168}$ MSE	$\texttt{ETTh2}_{168}$ MAE
Ordinal descriptions	4792	70.3%	0.52	0.55	0.59	0.83
Noisy descriptions	4813	71.2%	0.44	0.48	0.59	0.85
w/ annotated descriptions	**4897**	**71.4%**	**0.42**	0.49	**0.57**	**0.80**

Table 26: **Effect of Sensor Descriptions (TCN_{conv})**

Embedding Model	# Correct	Accuracy	$\texttt{ETTh1}_{168}$ MSE	$\texttt{ETTh1}_{168}$ MAE	$\texttt{ETTh2}_{168}$ MSE	$\texttt{ETTh2}_{168}$ MAE
mpnet	4893	71.3%	**0.41**	**0.45**	0.63	0.85
minilm	**4902**	**72%**	0.43	0.47	0.65	0.95
nomic	4897	71.4%	0.42	0.49	**0.57**	**0.80**

Table 27: **Effect of Text Embedding Models**

Configuration	# Correct	Accuracy
w/ additive embeddings (`all layers`)	4095	60.5%
w/ additive embeddings (`layer 0`)	4375	63.3%
$\mathbf{\Delta + G}$	**4897**	**71.4%**

Table 28: **Alternative Multimodal Approaches: additive vs. custom attention**

As shown in Table 26, perturbing or replacing channel descriptions leads to a moderate performance drop; however, the model remains reasonably robust to noisy descriptions. Table 27 reveals varied and inconclusive trends across different textual embedding models, with each performing well on distinct metrics. Finally, Table 28 suggests that the naive integration of text embeddings into the architecture is overly heavy-handed and results in performance degradation, particularly when embeddings are

injected directly into all layers of the encoder stack. These findings indicate that incorporating channel descriptions into a time series transformer requires greater nuance and more principled design choices.

L.1 Component Decomposition

To disentangle the contributions of the conditioning architecture, the JEPA objective, and text semantics, we conduct additional controlled ablations using the same 200-epoch protocol as the base model. Each ablation modifies exactly one factor while holding all others fixed. Results are reported in Table 29.

Configuration	Accuracy	# Correct	ETTh1 MSE	ETTh2 MSE	Avg MSE
Baseline (full CHARM)	**66.7%**	**6148**	0.4316	**0.3917**	**0.4117**
E1: No regularization (R_1+$R_2 = 0$)	64.3%	5986	**0.4288**	0.3947	0.4117
E2a: Random embeddings (same params)	66.0%	6194	0.4365	0.3946	0.4155
E2b: Shuffled descriptions	64.1%	5883	0.4311	0.3963	0.4137
E3: Reconstruction-only (no JEPA)	62.3%	5889	0.4322	0.3984	0.4153
E9: Channel-ID vectors (no text)	68.3%	6282	0.4306	0.3969	0.4138

Table 29: **Component decomposition.** Each row modifies exactly one factor relative to the baseline. E1 removes the regularization losses (R_1, R_2). E2a replaces text embeddings with fixed random $\mathcal{N}(0, 1)$ vectors (same architecture, same parameter count). E2b shuffles real text embeddings across channels within each batch (same distribution, wrong correspondence). E3 replaces the JEPA latent-prediction loss with a quantile reconstruction loss (same architecture and parameters). E9 replaces text embeddings with fixed per-channel-index vectors (same conditioning mechanism, zero text semantics).

These results decompose the overall contribution as follows:

- **JEPA objective** (-4.4 pp classification, E3): The largest single-factor degradation. Replacing latent prediction with reconstruction produces the lowest classification accuracy across all ablations, confirming that predicting in latent space is particularly well-suited for transferable sensor representations.
- **Conditioning mechanism** ($+5.0$ pp over naive fusion): Channel-ID vectors through the conditioning mechanism (E9, 68.3%) substantially outperform naive additive text fusion (63.3%, Table 28), even without any text semantics. The mechanism itself—not just its input—is the primary architectural driver.
- **Regularization** (-2.4 pp, E1): Removing the gate sparsity (R_1) and delta smoothness (R_2) losses degrades classification while leaving forecasting unchanged, confirming that regularization promotes structured inter-channel representations that specifically benefit classification.
- **Text semantics**: Random embeddings (E2a) degrade forecasting MSE by 0.9% while maintaining classification, suggesting text content matters more for per-channel temporal adaptation. Shuffled embeddings (E2b, -2.6 pp) perform worse than random (-0.7 pp), demonstrating the model actively exploits the correct channel–text correspondence rather than merely benefiting from added parameters.
- **Channel differentiation vs. text**: E9 (channel-ID, no text) achieves the highest classification accuracy (68.3%) but slightly worse forecasting (0.4138 vs. 0.4117 avg MSE). Channel-ID vectors provide stable per-channel identity within a dataset but are inherently non-transferable to unseen sensor configurations. Text embeddings carry portable semantics that enable cross-dataset transfer, a property exercised in our held-out evaluations (Exchange Rate, UCI Hydraulics).

M Embedding Visualizations

M.1 Emergence of intra-class label separation

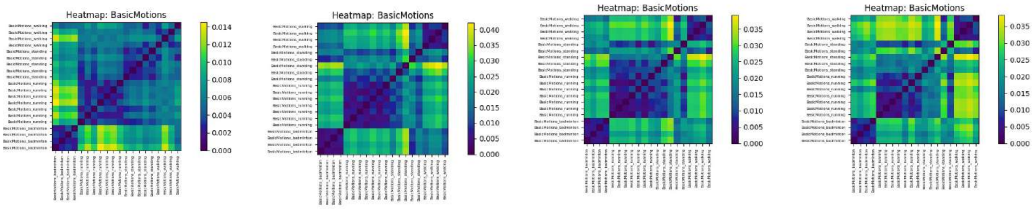
To analyze how our model's embeddings evolve over training, we plot similarity heatmaps of our embeddings on labelled datasets.

We first obtain embeddings for a dataset by sampling a subset (approximately 50 samples) of the full dataset, while ensuring we have full label coverage. Given this embedding matrix $\mathbf{Z} \in \mathbb{R}^{N_t \times T \times C \times H}$, we obtain our mean-pooled embeddings $\bar{\mathbf{Z}} \in \mathbb{R}^{N_t \times H}$ by averaging over the channel and time dimension.

Finally, the $N_t \times N_t$ similarity matrix, $\mathbf{S}$ is obtained as follows:

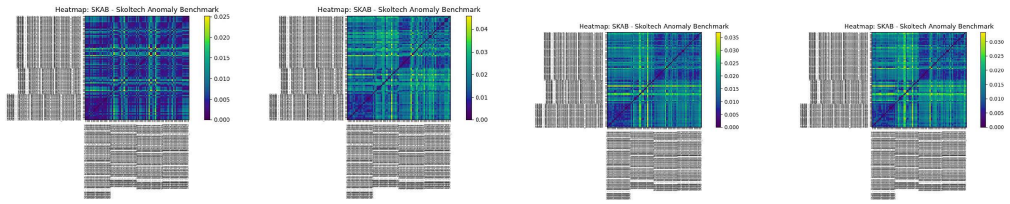
$$\mathbf{S}_{i,j} = ||\mathbf{Z}_{i,:} - \mathbf{Z}_{j,:}||_1 \tag{11}$$

We visualize the similarity matrix as a heatmap, as shown in Figures 23 to 25, and observe the emergence of structured clusters aligned with class labels. As training progresses, a block-diagonal structure[7]becomes increasingly prominent, wherein samples sharing the same label exhibit reduced Euclidean separation compared to those from different classes. This pattern reflects a progressive tightening of intra-class representations, indicative of improved semantic organization in the learned embedding space.



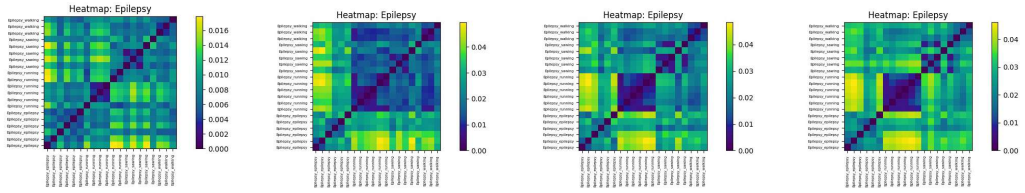
((a)) Epoch 0 ((b)) Epoch 3 ((c)) Epoch 6 ((d)) Epoch 9

Figure 23: Evolution of `BasicMotions` similarity heatmaps over training epochs



((a)) Epoch 0 ((b)) Epoch 3 ((c)) Epoch 6 ((d)) Epoch 9

Figure 24: Evolution of `Skoltech Anomaly Benchmark` similarity heatmaps over training epochs



((a)) Epoch 0 ((b)) Epoch 3 ((c)) Epoch 6 ((d)) Epoch 9

Figure 25: Evolution of `Epilepsy` similarity heatmaps over training epochs

[7]The heatmaps have a block structure because the labels are grouped together on each axis before plotting.

M.2 Evolution of Channel Gates

In this section we aim to visualize how channel gates, as defined in Paragraph Section 2.1.1, evolve over the course of training our model. We plot the gating matrix, $\mathbf{G}_d$, for each dataset for different checkpoints.

As illustrated in Figure 27, the inter-channel gating mechanism enables the model to dynamically modulate attention across channels, selectively emphasizing or suppressing information based on configurations that minimize the self-supervised learning (SSL) loss. We also empirically observe that the regularization loss begins to increase after an initial decline which suggests that after a certain point the model's embeddings require richer contextual information to continue improving.

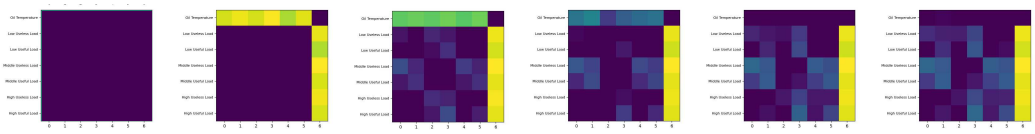


Figure 26: Evolution of Channel Gates for the `ETT` Dataset

The `ETT` dataset introduced by (Zhou et al., 2021a) comprises seven variables: `High Useful Load`, `Middle Useful Load`, `Low Useful Load`, `High Useless Load`, `Middle Useless Load`, `Low Useless Load`, and `Oil Temperature`. Among these, `Oil Temperature` serves as the target variable, with the remaining six acting as input features. During training, we observe a notable evolution in the learned channel gating patterns. Initially, the `Oil Temperature` channel does not attend to any other inputs, as indicated by its high gating values across all dimensions in Figure 26. However, as training progresses, this channel begins to incorporate information from all other variables. Interestingly, this behavior is asymmetric: while the target channel attends to all input features, the reverse does not occur—the other channels do not attend to `Oil Temperature`. This asymmetry manifests as a distinctive row-column pattern in the gating matrix and aligns with the underlying data semantics, where the target variable is causally influenced by the independent variables but not vice versa. These observations suggest that introducing learnable gating mechanisms can reveal interpretable, directional dependencies between variables which also increases model interpretability.

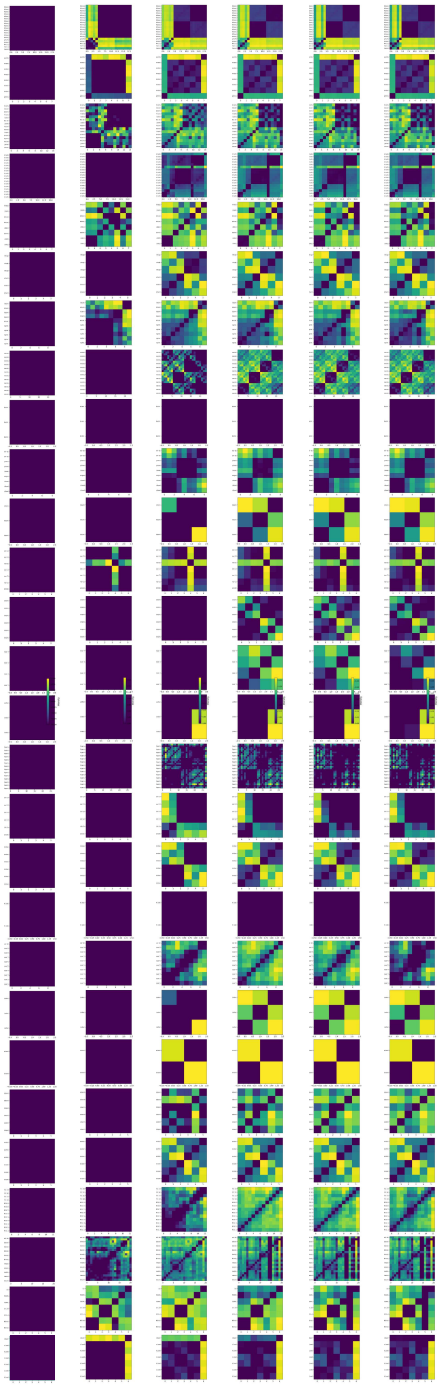


Figure 27: Evolution of inter channel gates during training. Checkpoints extracted at `epoch=0;step=49`, `epoch=0;step=499`, `epoch=0;step=999`, `epoch=2;step=49`, `epoch=6;step=49`, `epoch=8;step=49`.
Each row represents a particular dataset. Each column represents a sampled checkpoint as training progresses. Each heatmap represents $\mathbf{G}_d$ for a particular dataset, which is a $C \times C$ matrix with values in $[0, 1]$. Brighter colors on the heatmap represent **higher** gating values, i.e. decreased cross-channel interactions.