Supplementary Material

Paper Title: "**Short-Term Earthquake Forecasting via Self-Supervised Learning**"
Authors: **Yufeng Jiang, Zining Yu, and Haiyong Zheng, *Senior Member, IEEE***

A. Dataset

a) Data Source

The electromagnetic data is sourced from AETA (acoustic and electromagnetics to artificial intelligence) developed by the Key Laboratory of Integrated Microsystems of Peking University Shenzhen Graduate School. Our raw data is obtained from the AETA website (https://competition.aeta.io, latest access: May 14, 2023). Now, everyone who wishes to access the data needs to apply for higher permissions. The earthquake catalog is obtained from the China Earthquake Networks Center (https://news.ceic.ac.cn).

The data covers the geographical expanse in the southwestern region of China (latitude: 22°N–34°N, longitude: 98°E–107°E), spanning from January 1, 2017, to May 13, 2023. During this period, 39 earthquakes with magnitudes of 5 or higher occurred, as depicted in the spatial distribution shown in Figure A.

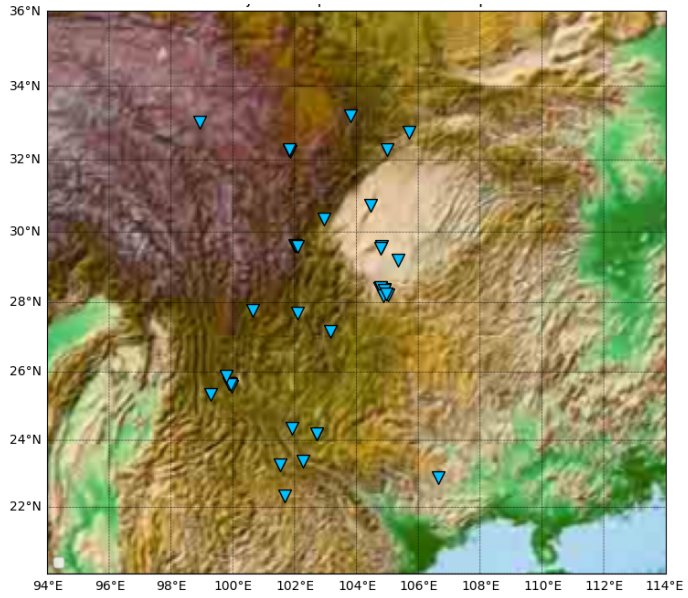

Figure A. Major earthquake distribution map.

b) Dataset Construction

First, we conduct data preprocessing, which includes station selection, data cleaning, missing data imputation, and data normalization. The relevant operations are described in the main file. Then, datasets are constructed separately for the pretext task and downstream task.

Pretext task. We set January 1, 2022, as the boundary. Data before this date is used to construct the training set, while data after is used to build the test set.

Given that the pretext task requires utilizing electromagnetic data from the current week to predict that of the following week, constructing the dataset necessitates providing data from the current week as input and data from the next week as the target. As the data is granular to ten-minute intervals, the data length for a week is $144 \times 7=1008$, resulting in the length of sliding window size being 1008. We employ three features obtained through Fourier transformation as the features. Therefore, the shape of each input or target is (1008, 3). As illustrated in Fig.3, the sliding step size for the training set is one day, equivalent to 144 data points, with adjacent samples overlapping. For the test set, the sliding step size is seven days, with no overlap between adjacent samples.

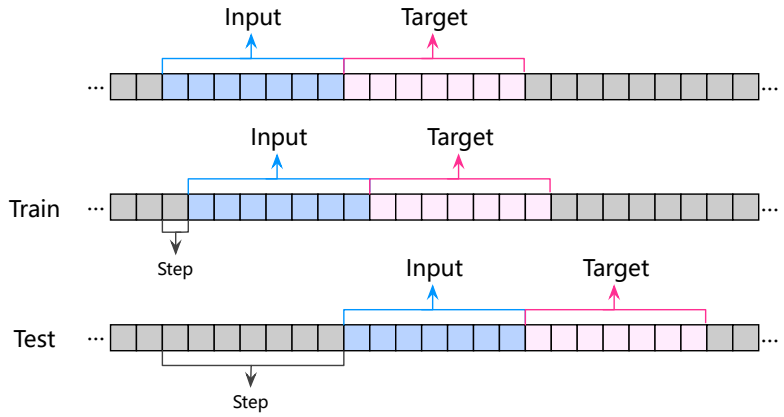

Figure B. Dataset construction for the pretext task.

Downstream task. Similarly, we use January 1, 2022, as the boundary, where data before this date are used for the training set and data after for the test set. Since the task requires predicting whether a five or higher magnitude earthquake will occur in the following week based on the current week's data, the data for the current week serves as input. In contrast, the earthquake occurrence in the subsequent week serves as the label. Both the earthquake magnitude and epicentral distance are considered when labeling, as detailed in Table A. For constructing the training set, the sliding window size remains 1008, with input data shape (1008, 3), and the label is the earthquake occurrence in the next week (0 for non-seismic, 1 for seismic), with a sliding step of 1 day. The sliding step is set to 7 days for constructing the test set.

Table A. Labeling method.

Magnitude	Epicenter Distance (km)	Label
[5, 6]	[0, 100]	1
[6, +∞)	[0, 200]	1
others	others	0

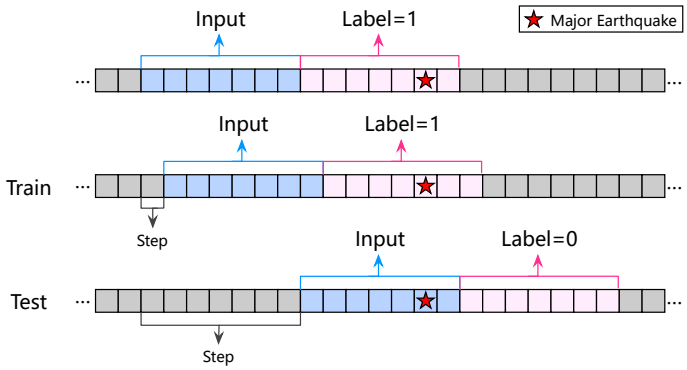

Figure C. Dataset construction for the downstream task.

B. Baselines

a) Machine Learning Models

Decision Tree (DT). Using the Gini impurity as the splitting criterion, and setting the maximum depth of the tree to 5.

Random Forest (RF). The forest consists of 10 decision trees, with a maximum depth set to 5 for each tree, still using the Gini index as the splitting criterion.

Support Vector Machine (SVM). Set the kernel type to "rbf", enable probability estimation, and limit the maximum number of iterations to 10000.

b) Deep Learning Models

FC. The shape of the input data is (B, 1008, 3), where B is the batch size, 1008 is the sequence length (one week), and 3 is the number of features. First, the input data is reshaped into (B, 1008×3) and then passed through a linear layer for classification.

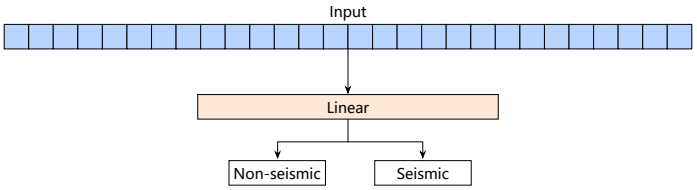

Figure D. The architecture of FC.

MLP. The shape of the input data is (B, 1008, 3). Similarly, it is reshaped into (B, 1008×3), then passed through two linear layers and a ReLU activation function for binary classification.

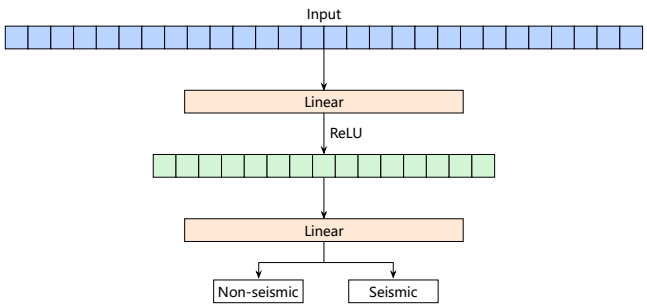

Figure E. The architecture of MLP.

BiLSTM. The shape of the input data is (B, 1008, 3). First, it passes through a BiLSTM with two hidden layers. The output from the last time step is then fed into an MLP for binary classification.

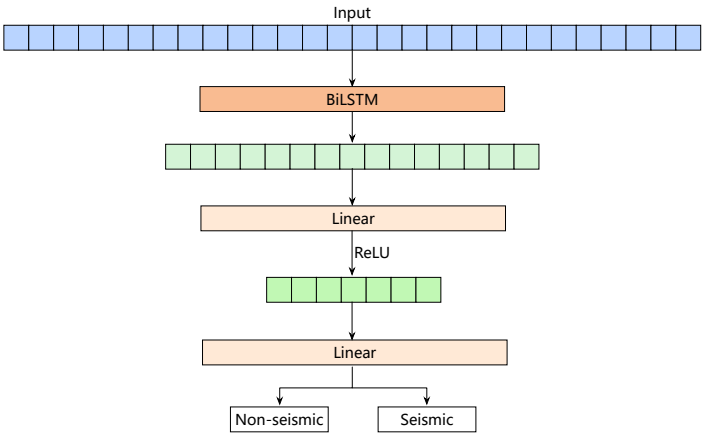

Figure F. The architecture of BiLSTM.

ViT. When ViT processes images, it splits them into multiple patches, each serving as a token. Each two-dimensional token is flattened into one-dimensional data and embedded into a vector. In this paper, we skip the patch-splitting process since the original data is one-dimensional. Instead, we treat every three adjacent data points as a token and directly use a 1D convolutional neural network with a kernel size of 3 to map each token into a vector. Next, we introduce a classification token (CLS_token) at the beginning of the vector sequence. This token plays a crucial role in guiding the model's classification process. Finally, we enhance the model's understanding of the sequence by adding positional encoding to each position. This step provides the model with vital information about the different positions in the sequence. The data then passes through the Transformer encoder. The output vector at the first position (CLS_token) is extracted and fed into an MLP for binary classification.

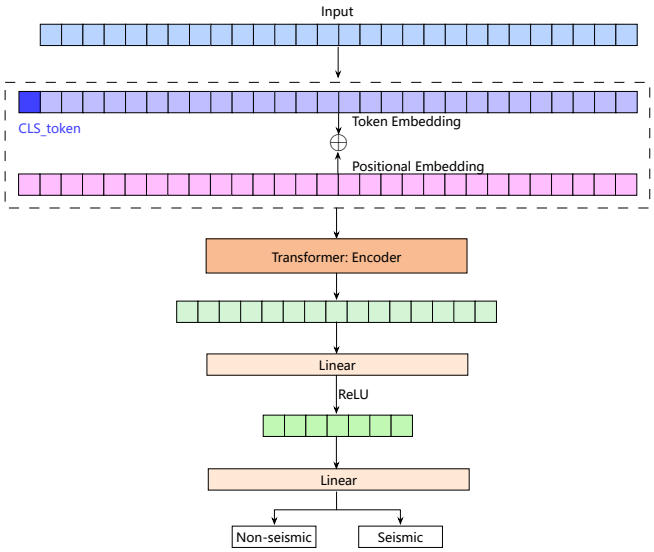

Figure G. The architecture of ViT.

AETA_CNN. We reproduced the AETA_CNN code based on the paper. First, the time series data is arranged in a two-dimensional format, with the input data having a shape of (B, 3, 144, 7), where B is the batch size, 3 is the number of features, 144 is the data per day, and 7 is the number of days. The data then sequentially passes through an Inception block consisting of 4 parallel paths, a down-sampling module composed of 4 sets of 2D convolutional layers, BatchNorm layers, and max-pooling layers, and a bottleneck block with residual connections. Finally, adaptive average pooling and a Linear layer are used for binary classification.

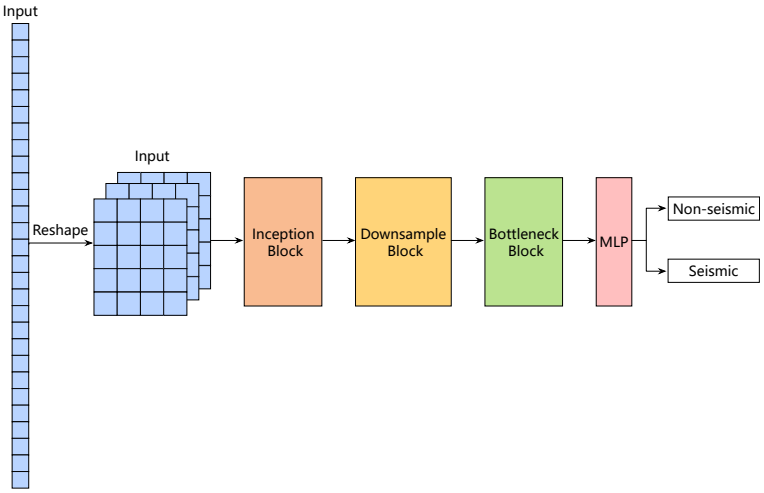

Figure H. The architecture of AETA_CNN.

ResNet18. The input data is reshaped from (B, 1008, 3) to (B, 3, 144, 7) and fed into ResNet18. The original ResNet18 outputs 1000 classes but has been modified to output two classes for this work.

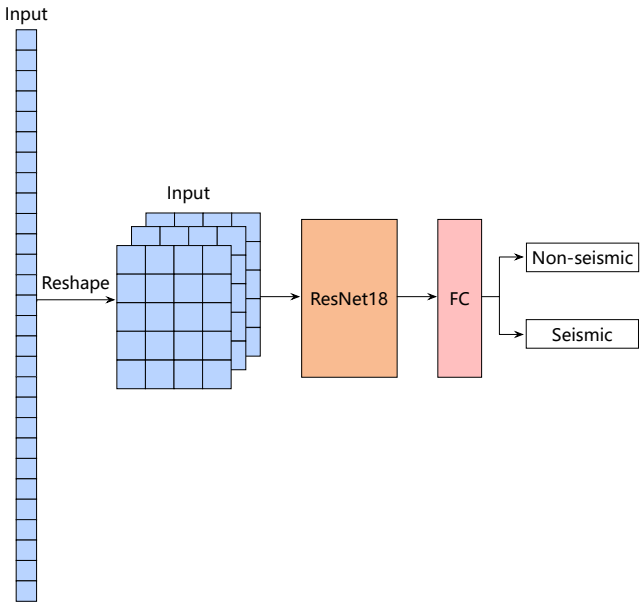

Figure I. The architecture of ResNet18.

Res_BiLSTM. First, reshape the input data from (B, 1008, 3) to (B, 3, 144, 7) and feed it into ResNet18. However, the adaptive average pooling and fully connected layers from ResNet18 should be removed, and the previously described BiLSTM is added to achieve binary classification.

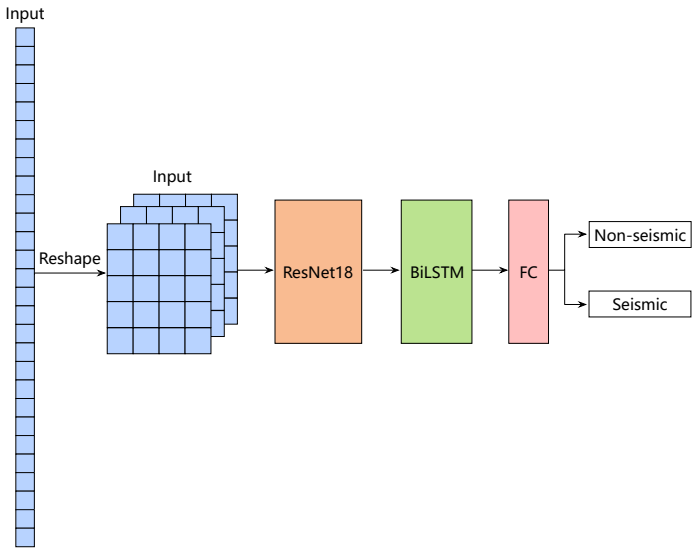

Figure J. The architecture of Res_BiLSTM.

C. Metrics

In a binary classification task, there are only Positive and Negative classes. Therefore, there are four possible outcomes for the predicted and true class, forming a confusion matrix (Table B):

- ✧ TP: True Positive, indicating that the predicted result and the true class are also positive, meaning the expected class matches the true class.
- ✧ FP: False Positive, indicating that the predicted result is positive, but the true class is negative, leading to a mismatch between the expected and true class.
- ✧ TN: True Negative, indicating that the predicted result and the true class are negative, meaning the expected class matches the true class.
- ✧ FN: False Negative, indicating that the predicted result is negative, but the true class is positive, meaning the expected class does not match the true class.

Table B. Confusion matrix.

Confusion Matrix		**True Class**	
		Positive	Negative
Predicted Class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

a) Accuracy

The proportion of all correctly predicted (positive and negative) cases out of the total.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad \text{(A)}$$

b) AUC

In machine learning and deep learning, many models output probabilities for classification problems, indicating the likelihood of belonging to a certain class. These probabilities need to be converted into class labels to calculate accuracy, which requires setting a threshold. Samples with probabilities greater than the threshold are assigned to one class, while those with probabilities less than the threshold are assigned to another. The choice of threshold directly impacts the accuracy calculation.

To determine the optimal threshold, we can consider each predicted probability value as a threshold, resulting in multiple sets of predictions. By calculating the False Positive Rate (FPR) and True Positive Rate (TPR) for each set of predictions, we obtain numerous coordinate points to plot the Receiver Operating Characteristic curve (ROC curve). The larger the Area Under the Curve (AUC), the better the model's classification performance.

AUC considers correct and incorrect classifications of positive and negative samples, making it effective even with slightly imbalanced datasets. That's why AUC is widely embraced for evaluating model performance in classification tasks.

$$FPR = \frac{FP}{FP+TN} \tag{B}$$

$$TPR = \frac{TP}{TP+FN} \tag{C}$$

$$AUC = \int_{x=0}^{1} TPR\left(FPR^{-1}(x)\right)dx = \int_{\infty}^{-\infty} TPR(T)FPR'(T)dT \tag{D}$$

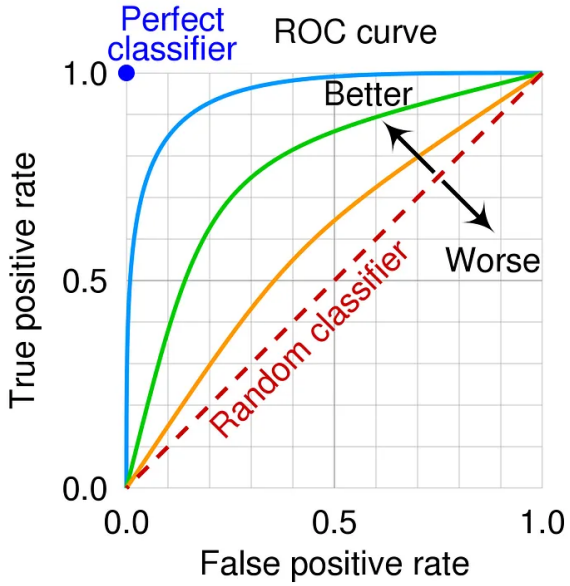

Figure K. The AUC for a "better" and "worse" classifier[1].

c) FNR

The proportion of all positive samples is predicted as negative (false negatives).

[1] This image is sourced from https://medium.com/@ilyurek/roc-curve-and-auc-evaluating-model-performance-c2178008b02.

$$FNR = \frac{FN}{FN+TP} \quad \text{(E)}$$

d) FPR

The proportion of all negative samples is predicted as positive (false positives).

$$FPR = \frac{FP}{FP+TN} \quad \text{(B)}$$

D. Experimental Details

a) Devices and Environment

Devices: 4 × NVIDIA GeForce RTX 3090.

Environment：

- ✧ python 3.7.16
- ✧ cudatoolkit 11.1.1
- ✧ torch 1.13.1
- ✧ torchvision 0.9.0
- ✧ numpy 1.21.5
- ✧ scikit-learn 1.0.2

b) Pretext Task

Unlike the calculation of training time (where the batch size is set to 4), the batch size is set to 16 during actual training. The model is trained for 30 epochs using the Adam optimizer with a learning rate of 0.0001 and an MSE loss function. All samples are used for training, with 172246 samples in the training set and 6710 samples in the test set.

c) Downstream task

The batch size is set to 4, and the model is trained for 100 epochs using Adam optimizer with a learning rate of 0.00001 and cross-entropy loss function. During training, all seismic samples are retained (a total of 684 samples), and undersampling is applied to non-seismic samples, randomly selecting 684 non-seismic samples. All test samples are used for evaluation, including 100 seismic samples and 7173 non-seismic samples.

E. Model Complexity and Computational Time

a) Model Complexity

We provide two metrics to evaluate the complexity of deep learning models: the parameter counts (Params) and floating-point operations (FLOPs).

Parameters refer to the quantity of learnable parameters in the model, typicallyweights and biases. More parameters generally indicate a more complex modelwith greater learning capacity.

FLOPs denote the number of floating-point calculations required for model inference or training, typically involving multiplication and addition operations. FLOPs directly reflect the computational cost of the model and serve as a measure of computational complexity.

b) Computational Time

Training time may be more critical in the early stages of model development. Because the model needs to undergo multiple training and tuning iterations, each consuming a significant amount of time. Therefore, we set the batch size for each deep learning method to 4 and calculate the training time for one iteration during training.

Inference time becomes more critical upon completion of model development. Shorter inference times mean faster access to prediction results. Therefore, we provide the inference time for a single sample as a reference.

Table C. Model complexity and computational time.

Methods	Params (M)	FLOPs (M)	Training time (s)	Inference time (s)
FC	0.006	0.006	0.002	0.0002
MLP	1.550	1.549	0.003	0.0003
BiLSTM	0.548	539.853	0.024	0.0014
ViT	13.407	13537.433	0.035	0.0040
AETA_CNN	2.475	702.465	0.015	0.0022
ResNet18	11.178	83.876	0.023	0.0033
Res_BiLSTM	12.246	89.172	0.026	0.0036
SSL-EF	11.317	10077.143	0.043	0.0091
	8.171	6098.207	0.047	0.0064

F. Ablation Study

This paper combines three embedding techniques for input data and conducts ablation experiments to determine the impact of each technique on the experimental results. There are three sets of experiments:

- ✧ Removing token embedding (denoted as w/o token embedding)
- ✧ Removing positional embedding (denoted as w/o positional embedding)
- ✧ Removing temporal embedding (denoted as w/o temporal embedding)

Each set of experiments is trained and tested three times and the average is taken as the final result.

The experimental results show that removing any of these three techniques will lead to a decrease in experimental results, indicating that all three are indispensable. These three embedding techniques encode the data from different perspectives and play crucial roles.

Table D. Quantitative results of the ablation studies on embedding techniques.

Ablated Models	AUC	Acc	FNR	FPR
w/o token embedding	0.614	0.505	0.487	0.495
w/o positional embedding	0.616	0.696	0.550	0.301
w/o temporal embedding	0.555	0.607	0.600	0.391

G. Visualization

Our task is to predict whether earthquakes of magnitude five or higher will occur in the coming week. Therefore, to evaluate the effectiveness of our trained model, we use

the test set for visualization to determine whether the learned representation distinguishes major earthquakes.

Specifically, each sample in the test set is fed into a pre-trained encoder to obtain its representation, each with a shape of (504, 512). We select an equal number of seismic and non-seismic representations and then calculate the cosine similarity between all representations to create a heatmap (Fig.13). In Fig.13, the top-left area depicts the similarity between seismic representations, the bottom-right area represents non-seismic similarities, and the top-right and bottom-left areas depict similarities between seismic and non-seismic representations. The heatmap clearly shows that representations within the same class exhibit high similarity. In contrast, those from different classes show lower similarity, indicating that the representations learned through SSL-EF can effectively distinguish between seismic and non-seismic events.

However, our method still has limitations, as it cannot accurately distinguish all samples in the test set. As shown in Fig.14, there are cases where representations within the same class have low similarity, which correlates with our method not achieving state-of-the-art accuracy. This suggests that while self-supervised learning for short-term earthquake forecasting shows promise, further research is needed to develop more effective representations and achieve more accurate predictions.

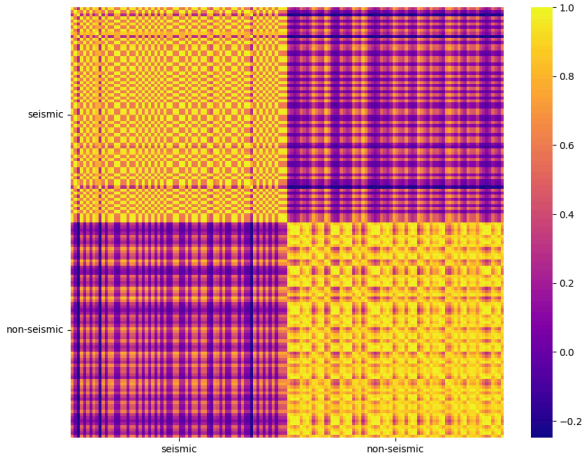

Figure L. Similarity heatmap of representations.

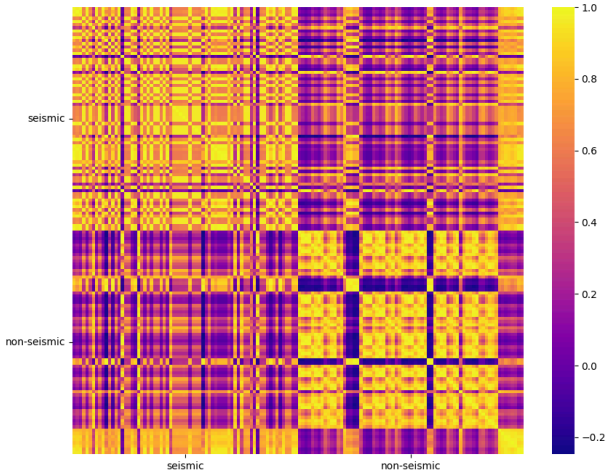

Figure M. Visualization of learned representations.