Deep Learning Methods for Movement Classification and Artifact Correction for Reliable Brain-Computer Interfaces

Annie Taylor
UC San Francisco
annie.taylor@ucsf.edu

Isadora White
UC Berkeley
isadoracw@berkeley.edu

Zheyuan Hu
UC Berkeley
huzheyuan@berkeley.edu

Shiangyi Lin
UC Berkeley
shiangyi.andrea.lin@berkeley.edu

***Abstract*—Stable neuroprosthetic control of robotic devices is a central goal of brain-computer interface (BCI) research. Electrocorticography (ECoG) is a widely used neural recording technique which isolates the brain surface voltages with sufficient complexity for BCIs to decode movements and instruct robotic control while offering improved reliability and long-term stability over other recording techniques [1]. Despite these successes, there still exist barriers to long-term BCI use in a clinical setting, as these devices may suffer from drift due to physical movement as well as gradual changes in the neural representation of movement [2]. Computational approaches to address these issues with signal stability could significantly improve the translational potential of ECoG-based BCIs. Here, we describe the first use of transformer-based models to decode movement from human cortical data recorded with ECoG. We demonstrate that (1) transformer-based autoencoders can reproduce denoised signals and (2) a transformer-based classifier performs well at movement classification tasks. In addition, we demonstrate that our MLP model which takes into account both temporal and spatial aspects of ECoG signals can perform better than the current State of the Art on the unfiltered data.**

***Index terms*—Artifact correction, autoencoders, brain-computer interfaces (BCIs), transformers.**

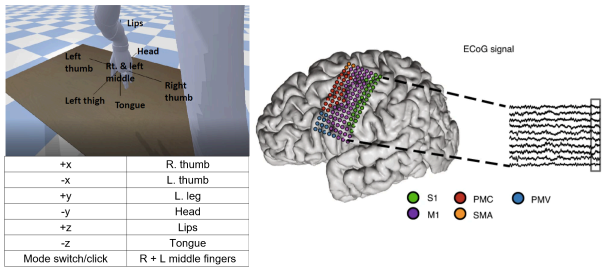
I. Introduction

BCIs have tremendous potential to improve the autonomy of individuals with motor-impairments, largely through motor and speech prostheses Existing BCI devices employ a variety of interfaces, from single unit spike-based recordings (microscale, e.g. Utah arrays), to local-field potentials (LFPs) recorded from the surface of the brain (mesoscale, e.g. ECoG) and even LFPs recorded from the skull surface (macroscale, e.g. electroencephalography or EEG) [3]. Larger scale recordings are typically less invasive and more stable over time, while more local recordings typically encode much richer activity, allowing for more precise movement decoding [1]. Mesoscale recording approaches such as ECoG are particularly promising for BCI applications [2].

While ECoG recordings are more stable than microscale BCIs, long-term relability is still a barrier to clinical use and user autonomy. For motor-impaired individuals to practically use BCI, changes in the physical connection or decoder must be infrequent. Even for ECoG devices, signal instability and drift require technicians to frequently disconnect and reconnect devices and recalibrate decoding models. The primary forms of instability observed in ECoG signals are electrode drift and representational drift. Electrode drift produces sharp, nonstationary changes in the recorded signal, which can interfere with the decoding process. Representational drift is a gradual change in the neural activity underlying movement, occurring over days to weeks, which necessitates recalibration of the decoder. We demonstrate a variety of computational approaches to correcting artifacts associated with electrode drift and suggest that extended applications of our models could also be useful for combatting representational drift.

Prior work includes a non-deep learning model Kalman-filter, applied to a similar ECoG dataset [2]; and deep learning methods such as LSTM applied to an augmented version of the dataset we used in this project (unpublished). A major contribution of our work is the application of transformers for BCI use. Transformer-based models have been tested on EEG recordings with notable success, generally outperforming other architectures on similar tasks and datasets [4], [5], [6] These results suggest that transformers are an ideal architecture for decoding neural signals, yet to-date there is no published literature using transformer-based approaches for microscale or mesoscale BCI devices. Micro to mesoscale recordings are much more broadly used for BCI, as prior work has demonstrated that local signals are critical for decoding neural signals associated with movement [3]. Here, we apply transformers to ECoG recordings from human sensorimotor cortex for the first time.

II. Materials and Methods



+x	R. thumb
-x	L. thumb
+y	L. leg
-y	Head
+z	Lips
-z	Tongue
Mode switch/click	R + L middle fingers

Figure 1: Left: screenshot of virtual robotic arm used in human experiments for cortical robotic control. Imagined movements and the corresponding degree-of-freedom are listed in underlying table, with each mapped to one of seven classes for decoding. Right: schematic of ECoG grid placement with example voltage traces.

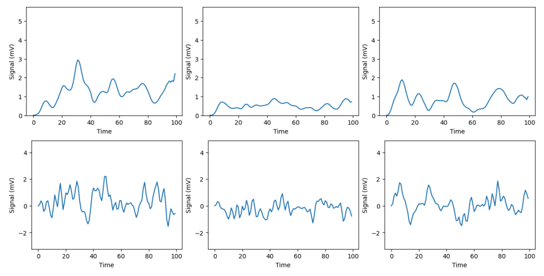


Figure 2: Exemplar traces (1s) of neural signals (mV) recorded from individual ECoG channels in both the high and low frequency bands. From left to right, we have channels 5, 51, 97; the first row is high frequency, and the second row is low frequency.

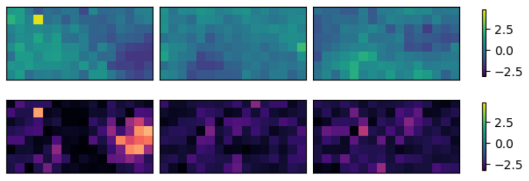


Figure 3: Spatial layout of signal from ECoG grid, plotted for 3 timesteps (0.05, 0.51, 0.97s from left to right). The first row is high frequency, and the second row is low frequency. The color intensities correspond to voltage in mV.

A. Dataset

In prior work, a 128-channel ECoG array was implanted in a single patient with severe motor impairment to enable recording of neural signals from sensory and motor cortices and raw data was referenced and filtered during acquisition [2]. We used a dataset with prefiltered ECoG signals paired with labels corresponding to one of seven classes mapped to an imagined movement. Labels were obtained by instructing the patient to imagine or attempt a specific movement (e.g. tongue, right thumb, left leg, etc.) and recording signal during the attempt. Instructions were provided for seven different movements and each movement class was then mapped to a degree of control ($\pm$ x, $\pm$ y, etc.) for a 7-DoF robotic arm (see Figure 1).

Each ECoG sample incorporates approximately 1 second of recorded data, downsampled from an acquisition rate 1kHz to 100Hz, and split into 100 equally spaced timesteps (Figure 2). For each timestep, there there were 256 recorded neural features. Recordings from each of the 128 ECoG channels were bandpass filtered into two features, one for low frequency ($<$ 25Hz) and the other high-gamma frequencyIz (>80Hz) band, for a total of 256 features as shown in Figure 3. The goal of classification is to map each ECoG signal to the correct movement class and use the decoding outputs for robotic control.

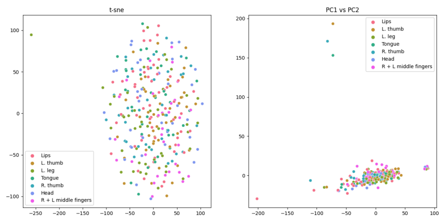


Figure 4: t-SNE and PCA of the first two components of sampled 300 datapoints. These plots shows that there are no simple pattern in our code.

B. Data Imputation and artifact correction

To develop models capable of handling signal artifacts in our data that disrupt decoding accuracy, we tested three primary strategies to remove these artifacts and impute the underlying signal. Examples of signal artifact are shown as a heatmap in Figure 5 and as timeseries in Figure 6. Going forward, we will refer to original data as raw data.

1. *Random noise imputation*: First, compute feature-wise mean $\vec{\mu}$ and overall standard deviation σ. Then, we computed updated feature-wise mean $\vec{\mu_*}$ and overall standard deviation σ_* using only the data within the range of $\vec{\mu} \pm 4\sigma$. For every data point outside of this range, we replace the data with random normal noise ~N$(\vec{\mu}, \sigma_*)$. Repeat this same process again with 10 instead of 4 to remove any potential remaining outliers. Figure 7 shows an example of random noise imputation, applied to the example artifact from Figure 5. This imputation method aims to remove artifact with noises within normal range so that the model

will not be able to overfit to artifact or rely too heavily on our artificial correction.

2. *Fourier transform imputation*: First, compute feature-wise mean $\vec{\mu}$ and feature-wise standard deviation $\vec{\sigma}$. We project the signal to fourier basis and remove frequencies with amplitude more than $\vec{\mu} + 4\vec{\sigma}$. This strategy is shown in Figure 8. We attempted this imputation method since we can treat signal wave functions and fourier transform would be able to decompose them.

3. *Simple clipping*: Here we apply Occam's Razor! After inspecting the data, we found that most of the signal is roughly mean-centered and in a range of ± 15. For instances where the signal exceeded ± 30, we clipped signal at ± 30 respetively. Clipped data is shown in Figure 9.

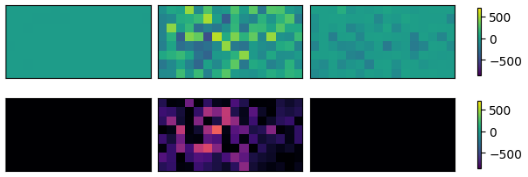


Figure 5: Representative raw signal with artifact, shown at three different timesteps. The artifact is apparent in the center image, producing a rescale in the colorbar that makes the signal at other timesteps difficult to visualize.

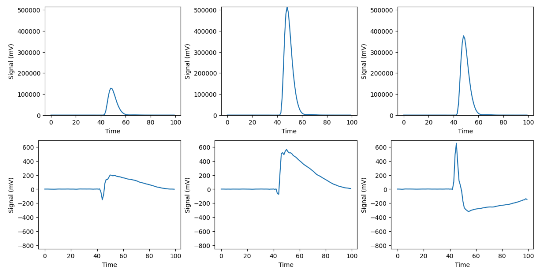


Figure 6: Representative raw signal with artifact, shown at three different channel and both frequencies. The artifact is apparent in the center of all images, where a large peak can be seen.

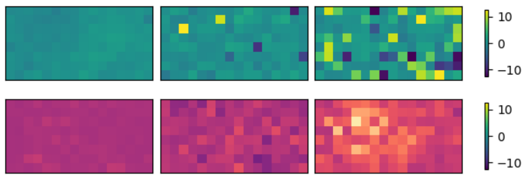


Figure 7: Signal from Figure 5 but imputed with random noise. After replacing the artifact with smaller random noise, smaller signals are visible in the image.

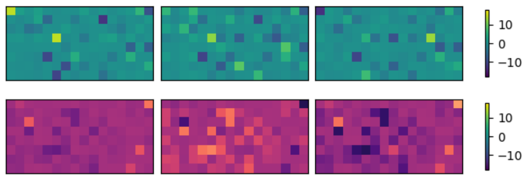


Figure 8: Signal from Figure 5 but imputed using fourier transformation. After removing the fitted frequency corresponding to the artifact, smaller signals are visible in the image.

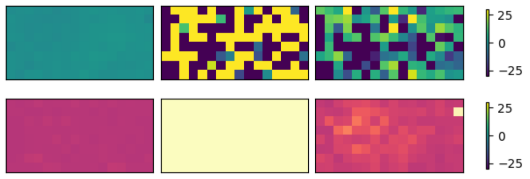


Figure 9: Signal from Figure 5 but clipped at ± 30mV. After clipping the signal, smaller signals are visible in the image.

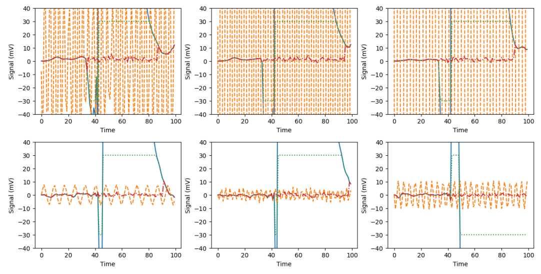


Figure 10: Overlay signal of raw and imputed signal of Figure 6, zoom in between [−40, 40]. Blue solid line denotes the raw signal, orange dash line denotes the random noise imputation, green dot line denotes the fourier transform imputation, and red dashdot line denotes the clipped signal respectively.

C. *Model Architectures*

1) *Transformer Autoencoder:*

Our autoencoder model uses an encoder-decoder Transformer architecture with multi-head self-attention and feed-forward networks for each layer [7]. The model is trained with reconstruction loss

$$\frac{1}{n}\sum_{i=1}^{n}\left(\|x_i - \widehat{x}_i\|_2^2\right) \tag{1}$$

Both the encoder and decoder employ 8 self-attention heads ($d_k = 512$) with fully-connected feedforward layers ($d_{\text{feedforward}} = 2048$) for 6 total layers. Preceding the encoder, we use a linear embedding layer between the input and the first encoding layer. To incorporate sequential inductive bias into the model, we add positional encodings (corresponding to the time dimension of our dataset) directly to the input embedding. Encoder outputs were passed directly to the decoder. Finally, the ouput layer in our autoencoder was a simple linear layer, projecting decoder output back to the original sample dimenisons ($512 \rightarrow 256$). The number of parameters are summarized in Table 1.

2) *Basic Classifier:*

We have build a simple classifier head that can accepts raw data, imputed data, or encoded data generated by our autoencoder as inputs.

Version 1: In this classifier, we performed a global average pooling across the time steps followed by a linear classification layer to output logits of the seven classes' preditions.

Version 2: We first apply a 1x1 convolution to reduce each channel's size. Then, we performed global average pooling across the timesteps and global average pooling across the channels seperately. The results are concatenated the results together and passed into a linear layer to output logits of the seven classes' preditions.

Version 3: The third version is similar to the second version with several additional layers. The concatenated global averages are passed into a linear projection layer, a Tanh activation before, and a dropout layer in the order before the final linear classification layer.

3) *Transformer Classifier:*

Transformer is a natural extension to the baseline LSTM model. Our transformer classifier employs a similar architecture to our autoencoder, but with the transformer decoding layer replaced by a single output layer that performs pooling over the feature dimension of the final encoder output, followed by a ReLU activation layer, layer normalization ($\varepsilon = 1e-5$), dropout ($p = 0.1$), and a single linear layer for classification. For classification, we tried both passing full-dimension output encodings to the classification layer as well as including a class token in our embedding and only passing the encoded class token to the classification layer. In some cases we also added causal masking during the forward pass, forcing the model to generate predictions only with earlier features in the sequence. The model is trained with cross-entropy loss:

$$-\frac{1}{n}\sum_{i=1}^{n}(p(x_i)(\log p(x_i))) \tag{2}$$

We include important architecture choices for the transfromer-based autoencoder and classifier in Table 1.

4) *LSTM Classifier (Baseline):*

Long Short Term Memory (LSTM) was used for the classification of robotic actions in an unpublished manuscript. A LSTM architecture incorporates state in two different ways: the long-term memory and short-term memory states. The learn gate determines which parts of short-term memory are relevant and the forget gate determines which parts of the long term memory state should be forgotten [8]. LSTM is a suitable model for our task becuase its architecture is designed to capture long-term dependencies in sequential data while avoiding the vanishing gradient problem. We trained this model to the imputed data we generated above to create a baseline comparison. Due to the confidential nature of the unpublished work, we will only report the results and not the details of implementaion and hyperameter choices.

annie check for the best hyper parameter

	Transformer Autoencoder	**Transformer Classifier**
Optimizer	AdamW	AdamW
d_k	512	512
n_l	6	6
n_h	8	4
d_o	2048	2048
learning rate η	$1e-4$	$1e-4$
weight decay γ	0.01	0.005
epochs	100	300

Table 1: Hyperparmeters used for transfromer autoencoder and transformer classifier.

III. Results

Next, for each of the models we designed, we tried different training approaches using combinations of the raw, imputed, and encoded datasets as inputs. We reported the best result among the hyperparameters we attempted in the section below; the details of hyperparameter tuning can be found in the appendix

A. *Autoencoder Experiments*

We ran four main experiments with the autoencoder model. In the first experiment, the model optimizes to reconstruct the original raw inputs. In the other three experiments, the model is trained to reconstruct the imputed samples corresponding to the raw data as inputs. The latter ones are intended to test the hypothesis that transformer autoencoders can be trained to perform artifact corrections. For the results shown in Table 2, we used the same set of learning parameters described in Table 1. We have included the results with different hyperparameters in the appendix.

The training loss is the MSE reconstruction loss computed as

$$\frac{1}{n}\sum_{i=1}^{n}\left(\|x-\hat{x}\|_2^2\right) \tag{3}$$

where x is the reconstruction target and $\hat{x}$ is the decoder's output. At evaluation time, reconstruction loss is measured between the model's outputs and the raw inputs. We exclude the NaN, losses less than 1 or greater than 99 percentile losses in evaluation as there are extremely large values in the raw inputs that are excerbated by the MSE Loss and would dominate the MSE Loss if included.

Input	Reconstruction Target	Training Loss	Eval Loss
Raw	Raw	43409.10	19811.73
Raw	Clipped	336.86	**351.96**
Raw	Random Noise	**107.69**	1026.93
Raw	Fourier	3380.18	198.89

Table 2: Losses for autoencoder variations trained with raw inputs and imputed data as reconstruction targets compared to raw data as target. The clipped imputation as reconstruction target achieves the best validation loss.

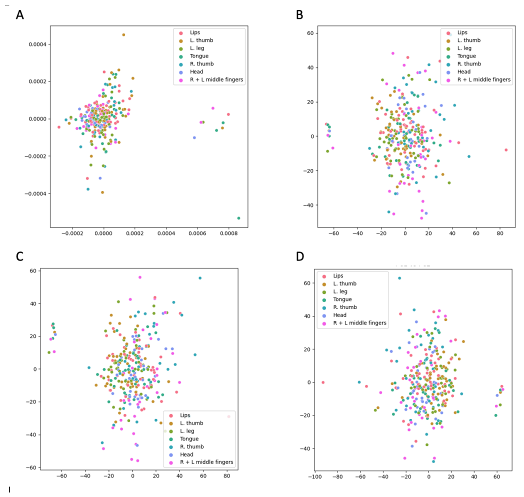


Figure 11: PC1 and PC2 of autoencoder embeddings trained with different sets of inputs (x) and template samples ($\hat{x}$); (A) raw to raw, (B) raw to clipped, (C) raw to fourier, (D) and raw to random noise imputation. Note that the points are much more evenly spread of then the PCA of the raw data Figure 4, suggesting our autoencoder is learning to remove some outliers.

B. *Classifier Experiments*

To test performance of our classifiers, we ran a series of experiments training and testing models with raw data, each data imputation approach as we did with the autoencoder, and the encoder output from our autoencoder before. In addition, we considered the impact of causal masking on decoding accuracy and the impact of appending a classification token to the input. Details of hyperparameters tuning can be found in the appendix. In summary, we found that including classification token and causal masking were very crucial for improving classification accuracy and improving training stability. When we train a model without a classification token, we found that training was very unstable and unpredictable. We believe that this is because the classification token allowed the model to extract more meaningful information for classification.

Additionally, we evaluated classifier designs using the output from a frozen autoencoder's encoder trained with raw-to-clipped data. This choice is based on the autoencoder performance analysis from previous part. The three different classifier models are described in the model architectures section. We report that the Basic v3 model works the best as classification heads, using either the autoencoder's encoder output or directly training on the raw inputs. Notably, with the best classifiation head design, training on autoencoder's encoder outputs outperforms the raw data inputs by almost 2%. We hypothesize that the classifier head benefits from the denoising effect from our autoencoder, such that outliers in the inputs can be classified more accurately. Results are summarized in Table 3.

Classifier model	Input	Accuracy	Validation Loss
LSTM	Raw	74.78%	577.275
LSTM	Clipped	75.54%	572.005
LSTM	Random	74.26%	583.484
LSTM	Fourier Transform	76.66%	562.507
Transformer	Raw	**78.12%**	**0.6949**
Transformer	Clipped	76.36%	0.7196
Transformer	Random	75.65%	0.7144
Transformer	Fourier Transform	75.85%	0.7129
Basic v1	Clipped Encoder	57.72%	1.182
Basic v2	Clipped Encoder	74.21%	0.7578
Basic v3	Clipped Encoder	**78.58%**	**0.6623**
Basic v3	Raw	76.86%	0.6735

Table 3: Accuracy and final losses for decoders trained and tested with different types of input data.

IV. Discussion

Accurate and stable decoding frameworks are essential for the development of translational BCI. Our initial work demonstrates that transformer based classification and artifact correction strategies could be effective in BCI research, with the potential to significantly improve state-of-the-art decoding methods. We found that imputating the data in the three different methods described in the paper, was able to significantly reduce the reconstruction loss for the autoencoder. In particular, the clipped data imputation performed the best and that the autoencoder filtering from raw data to clipped data was able to remove some of the outliers using the first two PCA dimensions.

The use of autoencoder are significant in two ways. First, by introducing artifact corrected targets we encourage the autoencoder to learn to ignore artifacts, producing the cleaner data that can be utilized by the downstream classifier. A more robust computational model will relax the physical constraint associated with ECoG device, saving time for clinician to disconnect and reconnect the device. Secondly, learning the underlying pattern of ECoG data is itself an interesting question. By learning the underlying latent space, there is a potential to uncover the pattern of representational drift, which is an active area of research. Although we did not perform explicit analysis on this front, this project can serve as a starting point for more future work.

When creating our classifier for ECoG data we took many different approaches. Our best performance was obtained by freezing the encoder layer of the autoencoder on clipped data and performing linear probing to classify the robotic motions. The second best performance was obtained by a transformer classifier on the raw data. However, a simple linear model which combined both positional and temporal information was still able to match the performance of the state of the art LSTM classifier. Ultimately, we think that this shows that the most critical information for movement decoding in the ECoG signal is represented in the temporal and spatial characteristics of the data and thus high dimensional models like transformers might only bring limited benefits to standard decoding methods that also incorporate spatiotemporal information.

V. Note for reviewers

Code: Our github repo is currently private as we are preparing for submission. It contains our artifact correction, models, training loop, and plottting code: https://github.com/icwhite/ecog_bci.

Data: Since we used clinical data, our dataset is confidential. If you are a reviewer and need data access, please contact the authors (annie.taylor@ucsf.edu) directly and we would be happy to share a subset of the data.

References

[1] Chao, "Long-term asynchronous decoding of arm motion using electrocorticographic signals in monkey", *Frontiers in Neuroengineering*, 2010, doi: 10.3389/fneng.2010.00003.

[2] D. B. Silversmith *et al.*, "Plug-and-play control of a brain–computer interface through neural map stabilization", *Nature Biotechnology*, no. 3, pp. 326–335, Mar. 2021, doi: 10.1038/s41587-020-0662-5.

[3] E. F. Chang, "Towards Large-Scale, Human-Based, Mesoscopic Neurotechnologies", *Neuron*, no. 1, pp. 68–78, Apr. 2015, doi: 10.1016/j.neuron.2015.03.037.

[4] D. Kostas, S. Aroca-Ouellette, and F. Rudzicz, "BENDR: Using Transformers and a Contrastive Self-Supervised Learning Task to Learn From Massive Amounts of EEG Data", *Frontiers in Human Neuroscience*, p. 653659, Jun. 2021, doi: 10.3389/fnhum.2021.653659.

[5] J. Xie *et al.*, "A Transformer-Based Approach Combining Deep Learning Network and Spatial-Temporal Information for Raw EEG Classification", *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, pp. 2126–2136, 2022, doi: 10.1109/TNSRE.2022.3194600.

[6] P. Deny and K. W. Choi, "Hierarchical Transformer for Brain Computer Interface", Gangwon, Korea, Republic of: IEEE, Feb. 2023, pp. 1–5. doi: 10.1109/BCI57258.2023.10078473.

[7] A. Vaswani *et al.*, "Attention Is All You Need", 2017, doi: 10.48550/ARXIV.1706.03762.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, no. 8, pp. 1735–1780, 1997.

VI. Appendix

AutoEncoder Hyperparameter Choice:

	Option 1	**Option 2**
Optimizer	AdamW	AdamW
d_k	**32**	**512**
n_l	6	6
n_h	8	4
d_o	2048	2048
learning rate η	$1e-4$	$1e-4$
weight decay γ	0.01	0.005
epochs	100	300
input	raw	clipped
target	raw	clipped
training loss	3772.83	336.86

We experimented two different latent representation sizes (d_k) for transformer AutoEncoder. In typical AutoEncoders, such as the ones for images, the latent representation size is smaller than the input dimensions, as only a subset of features is useful for downstream classification tasks. However, our input samples are high dimension time series data, reducing latent dimensionality could actually harm the performance. One way to rationalize this is to imagine classification as finding linearly seperable hyperplanes in the latent space. As we see in earlier sections of this report, the visualization of the low dimensional projection of our input data is clustered closely and noisy. So it makes more sense to project the inputs to a higher dimensions and hope it becomes more seperable in that space.

Transformer Classifier Hyperparameter Choice:

	Default Parameters
Optimizer	AdamW
d_k	512
n_l	6
n_h	8
d_o	2048
Dropout	None
Casual Masking	None
learning rate η	$1e-4$
weight decay γ	0.005
epochs	300
input	Raw
target	Class labels
Accuracy	75.44%

Table 4: Default Transformer Classifier Hyperparameters

Parameters changed	**Accuracy**
$d_k = 512$	75.05%
$d_k = 1024$	76.74%
$d_k = 2048$	73.99%
$n_l = 4$	75.93%
$n_l = 6$	75.04%
$n_l = 8$	76.42%
$n_h = 4$	77.11%
$n_h = 8$	74.02%
$d_o = 512$	75.32%
$d_o = 1024$	74.27%
$d_o = 2048$	76.22%
$d_o = 4096$	76.95

Table 5: Transformer Hyperparameters used for Tuning

In Table 4, we showed our default hyperparameters of the transformer classifier. We perform hyperparameter tuning by modifying one parameter at the time, and the result are displayed in Table 5. Each row in Table 5 coresponds to a model. The hyperparameters values correspond to the default values unless otherwise specified. For example, the first row we show the model with $d_k = 512$ with other default hyperparameters: AdamW optimizer, $n_l = 6$, $n_h = 8$, $d_o = 2048$, no dropout, no casual mask, $\eta = 1e-4$, $\gamma = 0.005$, 300 epochs, raw input. The accuracies are 75.05%, shown in the last column.