Achieving High-performance Cross-day Neural Decoding by Learning Channel-level Invariant Representations

Anonymous Authors[1]

Abstract

Brain-computer interfaces have shown great potential in motor and speech rehabilitation, but still suffer from low performance stability across days, mostly due to the instabilities in neural signals. These instabilities, partially caused by neuron deaths and electrode shifts, leading to channel-level variabilities among different recording days. Previous studies mostly focused on aligning multi-day neural signals of onto a low-dimensional latent manifold to reduce the variabilities, while faced with difficulties when neural signals exhibit significant drift. Here, we propose to learn a channel-level invariant neural representation to address the variabilities in channels across days. It contains a channel-rearrangement module to learn stable representations against electrode shifts, and a channel reconstruction module to handle the missing neurons. The proposed method achieved the state-of-the-art performance with cross-day decoding tasks over a month, on multiple benchmark BCI datasets. The proposed approach showed good generalization ability that can be incorporated to different neural networks.

1. Introduction

Brain-computer interfaces (BCIs) have paved a new way for motor and speech rehabilitation (Card et al., 2024; Metzger et al., 2023). However, the long-term stability of BCIs remains a critical problem due to the variability of neural signals. BCI systems commonly require frequent recalibration in use, which hinders the user experience (Ajiboye et al., 2017).

Existing studies addressed this problem from several aspects. The first group is the data alignment approaches, which aim to align the low-dimensional manifolds of neural signals of different days. Researchers have demonstrated that neural signals contain a consistent low-dimensional neural manifold over long periods of time (Gallego et al., 2017). Study (Degenhart et al., 2020) tries to directly align the manifolds' coordinate axes (the neural recording channels). However, it required that a subset of the channels should be stable across different days. Other studies based on domain adaptation (DA) approaches, which regard neural signals from different days as different domains and attempt to align the signals of these domains, including the ADAN (Farshchian et al., 2018) and Cycle-GAN (Ma et al., 2023) approaches. The second group attempts to improve the robustness of cross-day neural decoding by using more data. NDT2 (Ye et al., 2024) and POYO (Azabou et al., 2024) used a large set of neural data for pretraining to capture the pattern of variation between sessions. However, the channel-wise variability due to the instabilities of neurons has not been effectively handled by existing methods and still poses a challenge.

The channel-wise variability can stem from diverse reasons (Degenhart et al., 2020). First, the appearance of new neurons and the disappearance of existing neurons can happen across different experimental days (Figure 1a). Second, the movement of neurons or electrodes can also drift in the recorded neural signals at each channel (Figure 1b). The situations that cause channel-wise variability can be highly diverse, such that directly learning a channel-wise invariant neural representation can be a challenging problem. To directly address this issue, we propose a **C**hannel **R**earrangement and **R**econstruction **L**earning framework (CRRL). It contains two main modules: 1) a channel rearrangement module to deal with situations such as new coming neurons explicitly and drifts in neural signals, and a channel reconstruction module to handle situations such as missing neurons. We found that the two operations can highly enhance the channel-level consistency across different days, facilitating a channel-invariant neural representation and improving the neural decoding performance.

We evaluate the proposed method on both simulation datasets with diverse channel-wise variabilities and multiple neural signal datasets with motor, handwriting and speech decoding tasks. With the neural decoding tasks, we evalu-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

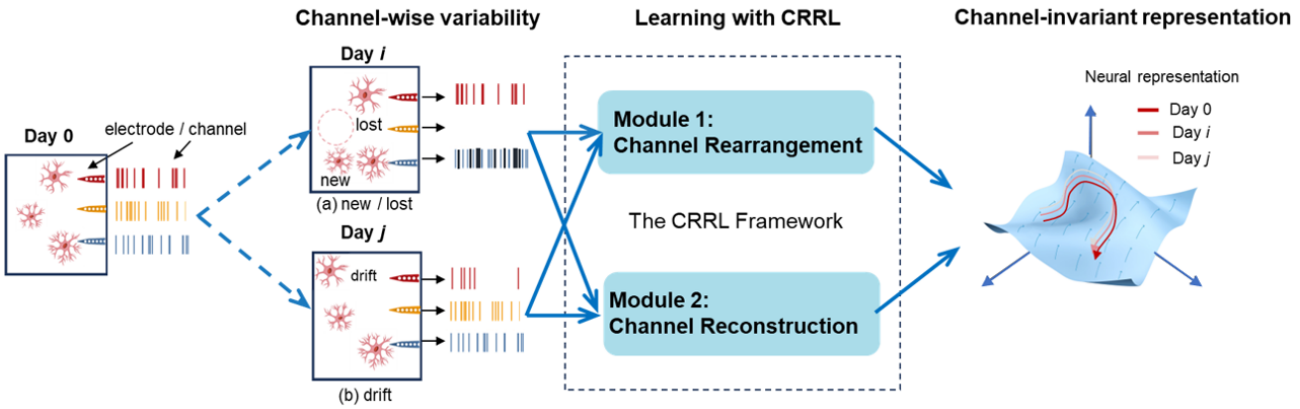

Figure 1. We plot two types of channel-wise variability in neural signals: (a) caused by new or lost neurons. (b) caused by drifted neurons. Our two-module CRRL framework can obtain the channel-invariant representations across days

ate the neural decoding performance of our approach with long-term neural signals with time spans over two months. Experimental results demonstrate that our approach achieves stable decoding performance compared with existing studies, especially for a long period. Moreover, our approach can be incorporated with other methods as a plug-in module, to enhance the neural recording performance and robustness across days.

2. Method

The factors causing channel-wise variability can be summarized into two categories: 1) new or lost neurons, and 2) drifted neurons. CRRL uses two modules to handle these two cases separately.

Specifically, in the rearrangement module, we use the permutation matrix obtained by our permutation network to rearrange the signals of day k, and the training objective is to maximize the similarity between each channel on day k and day 0. In the reconstruction module, we employ a channel-based Vector Quantized Variational Autoencoder (VQ-VAE) (Van Den Oord et al., 2017) with randomly masked signals of day 0 to achieve the reconstruction of channel signals and frequency domain features. During the evaluation, we used the data of day k+n as input and obtained the output sequentially through the rearrangement and reconstruction network. Then, we use the output signal to perform downstream decoding tasks, where the decoder is trained by the signals of day 0. The pseudo-code is presented in Algorithms 1 and 2.

2.1. Rearrangement: Channel Permutation Network

Below, we first formulate the tasks of channel permutation and describe the structure of our permutation network. Next, we describe the process of the Sinkhorn-Knopp algorithm (Mena et al., 2018) to obtain differentiable approximations of permutation matrices. Finally, we introduce our loss function for our rearrangement training.

2.1.1. Problem Formulation

Given neural signal data from k days $\mathcal{D} = \{\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_{k-1}\}$, where $\mathbf{X}_d \in \mathbb{R}^{N\times C\times T}$ represents the data from day d, with N samples, C channels, and T time points, our goal is to find permutation matrices $\{\mathbf{M}_1, \ldots, \mathbf{M}_{k-1}\}$ that rearrange each day's channels with those of day 0. The rearranged data is given by:

$$\mathbf{X}'_d = \mathbf{M}_d^\top \mathbf{X}_d, \quad \text{for } d = 1, \ldots, k-1. \tag{1}$$

2.1.2. Predicting Permutation Matrices by Neural Network

To parameterize the permutation matrices in a differentiable manner, we employ a multi-layer perceptron (MLP) f_θ with shared parameters θ across all days. For each day d, we flatten the data along the channel and time dimensions to form the input to the MLP:

$$\mathbf{x}_d^{\text{flat}} = \text{Flatten}(\mathbf{X}_d) \in \mathbb{R}^{N\times(C\times T)}. \tag{2}$$

The MLP processes the flattened input to produce logits representing the unnormalized log probabilities for the permutation matrix $\mathbf{L}_d$:

$$\mathbf{L}_d = f_\theta(\mathbf{x}_d^{\text{flat}}), \quad \mathbf{L}_d \in \mathbb{R}^{N\times C\times C}. \tag{3}$$

2.1.3. Sinkhorn Algorithm for Doubly Stochastic Approximation

Permutation matrices are special cases of doubly stochastic matrices with binary entries. To obtain differentiable approximations of permutation matrices while enabling gradient-based optimization, we adopt the Sinkhorn-Knopp algorithm with a continuous relaxation strategy (Mena et al., 2018). This relaxation allows the model to learn permutation patterns in a probabilistic manner during training,

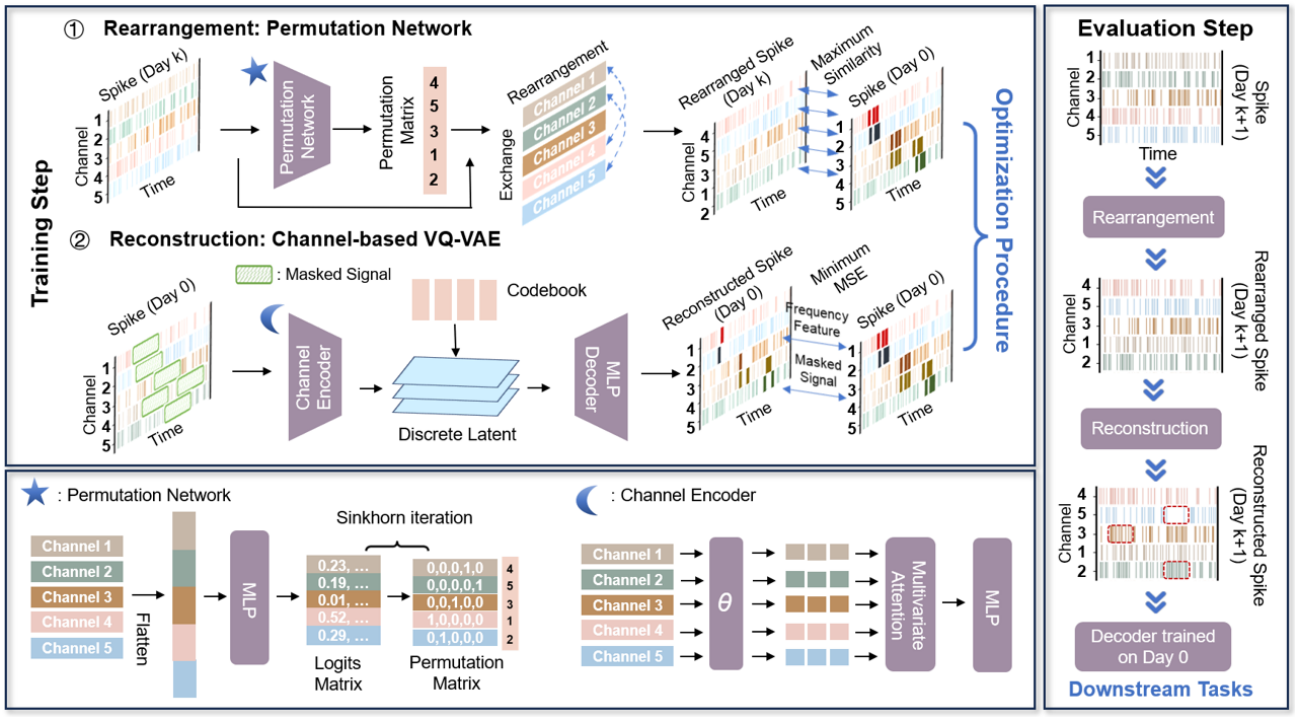

Figure 2. The framework of CRRL. Top: we plot our two-stage training process. Bottom: we plot the network structure of our permutation network and channel encoder. Right: we plot the evaluation process of CRRL, which needs a well-trained permutation network and channel-based VQ-VAE.

while recovering exact permutations via discretization at inference.

Continuous Relaxation via Gumbel-Sinkhorn To bridge the gap between discrete permutations and continuous optimization, we introduce a Gumbel noise perturbation into the logits $\mathbf{L}_d$, following the Gumbel-Sinkhorn framework (Jang et al., 2017a). The perturbed logits are defined as:

$$\tilde{\mathbf{L}}_d = \mathbf{L}_d + \gamma\epsilon, \quad \epsilon \sim \text{Gumbel}(0, 1), \tag{4}$$

where $\gamma > 0$ controls the noise intensity. This perturbation encourages exploration during training while maintaining differentiability. And the proof about this differentiability is detailed in Appendix.

The doubly stochastic matrix $\mathbf{M}_d$ is then computed by applying the **Sinkhorn operator** $\mathcal{S}$ to the perturbed logits:

$$\mathbf{M}_d = \mathcal{S}\left(\frac{\tilde{\mathbf{L}}_d}{\tau}\right), \tag{5}$$

where $\tau > 0$ is a temperature parameter. Lower τ sharpens the output distribution towards a permutation matrix, whereas higher τ results in a softer distribution.

Sinkhorn Operator with Iterative Normalization The Sinkhorn operator $\mathcal{S}$ iteratively normalizes the input matrix to satisfy row and column stochasticity constraints. Starting from the exponentiated logits:

$$\mathbf{K}_d^{(0)} = \exp\left(\frac{\tilde{\mathbf{L}}_d}{\tau}\right), \tag{6}$$

we perform alternating row and column normalizations for T_{sink} iterations:

$$\mathbf{K}_d^{(t)} = \mathbf{K}_d^{(t-1)} \oslash \left(\mathbf{K}_d^{(t-1)}\mathbf{1}_C\right) \quad \text{(Row norm)}, \tag{7}$$

$$\mathbf{K}_d^{(t)} = \mathbf{K}_d^{(t)} \oslash \left(\mathbf{1}_C^\top \mathbf{K}_d^{(t)}\right) \quad \text{(Column norm)}, \tag{8}$$

where $\oslash$ denotes element-wise division, and $\mathbf{1}_C$ is a vector of ones. After T_{sink} iterations, we obtain $\mathbf{M}_d = \mathbf{K}_d^{(T_{\text{sink}})}$.

Discretization via Hungarian Algorithm at Inference During training, $\mathbf{M}_d$ remains a continuous doubly stochastic matrix to preserve differentiability. At inference, we convert $\mathbf{M}_d$ to an exact permutation matrix $\mathbf{P}_d$ using the Hungarian algorithm (Kuhn, 1955):

2.1.4. Loss Functions For Rearrangement Training

To train the MLP to produce permutation matrices that effectively align the channels, we define a loss function com-

prising the negative Pearson correlation loss and an entropy regularization term.

Negative Pearson Correlation Loss For each task $t \in \{1, \ldots, T\}$ and channel $c \in \{1, \ldots, C\}$, we compute the average signal over all samples from day 0 belonging to task t:

$$\bar{\mathbf{x}}_{0,t,c} = \frac{1}{N_{0,t}} \sum_{n \in \mathcal{I}_{0,t}} \mathbf{x}_{0,n,c}, \tag{9}$$

where $N_{0,t}$ is the number of samples from day 0 belonging to task t, $\mathcal{I}_{0,t}$ is the index set of these samples, $\mathbf{x}_{0,n,c}$ is the signal for sample n and channel c from day 0 data.

Then, the negative Pearson correlation loss is defined as:

$$\mathcal{L}_{\text{corr},d} = -\frac{1}{NC} \sum_{n,c} \frac{\left\langle \mathbf{x}'_{d,n,c} - \overline{\mathbf{x}}'_{d,n,c},\ \bar{\mathbf{x}}_{0,y_n,c} - \overline{\bar{\mathbf{x}}}_{0,y_n,c} \right\rangle}{\left\| \mathbf{x}'_{d,n,c} - \overline{\mathbf{x}}'_{d,n,c} \right\|_2 \ \left\| \bar{\mathbf{x}}_{0,y_n,c} - \overline{\bar{\mathbf{x}}}_{0,y_n,c} \right\|_2}, \tag{10}$$

where $\mathbf{x}'_{d,n,c}$ is the signal for sample n and channel c from reordered day d data, y_n is the task label of sample n, $\bar{\mathbf{x}}_{0,y_n,c}$ is the average signal over all day 0 samples belonging to task y_n and channel c, $\overline{\mathbf{x}}'_{d,n,c}$ and $\overline{\bar{\mathbf{x}}}_{0,y_n,c}$ are the mean over time of $\mathbf{x}'_{d,n,c}$ and $\bar{\mathbf{x}}_{0,y_n,c}$, respectively.

The total correlation loss is then:

$$\mathcal{L}_{\text{corr}} = \sum_{d=1}^{k-1} \mathcal{L}_{\text{corr},d}. \tag{11}$$

Entropy Regularization To encourage the approximation matrices $\mathbf{M}_d$ to be close to true permutation matrices, we add an entropy regularization term:

$$\mathcal{L}_{\text{entropy},d} = -\frac{1}{N} \sum_{n}^{N} \sum_{i}^{C} \sum_{j}^{C} M_{d,n,i,j} \log M_{d,n,i,j}. \tag{12}$$

The total entropy regularization loss is:

$$\mathcal{L}_{\text{entropy}} = \sum_{d=1}^{k-1} \mathcal{L}_{\text{entropy},d}. \tag{13}$$

Total Loss for Stage One The overall loss function for stage one is:

$$\mathcal{L}_{\text{stage1}} = \mathcal{L}_{\text{corr}} + \lambda_{\text{entropy}} \mathcal{L}_{\text{entropy}}. \tag{14}$$

Algorithm 1 Rearrangement training

Require: Data $\mathcal{D} = \{\mathbf{X}_0, \ldots, \mathbf{X}_{k-1}\}$, initial temperature τ_{initial}, final temperature τ_{final}, decay rate γ, entropy coefficient λ_{entropy}, number of epochs E
Initialize: Network parameters θ, temperature $\tau \leftarrow \tau_{\text{initial}}$
for epoch = 1 **to** E **do**
  Sample minibatch $\{\mathbf{X}_0^{\text{batch}}, \ldots, \mathbf{X}_{k-1}^{\text{batch}}\}$
  for each day $d = 1$ to $k-1$ **do**
    Flatten input data: $\mathbf{x}_d^{\text{flat}} = \text{Flatten}(\mathbf{X}_d^{\text{batch}})$
    Compute logits: $\mathbf{L}_d = f_\theta(\mathbf{x}_d^{\text{flat}})$
    Introduce Gumbel noise: $\tilde{\mathbf{L}}_d = \mathbf{L}_d + \gamma\epsilon$
    Compute permutation matrix: $\mathbf{M}_d = \mathcal{S}(\tilde{\mathbf{L}}_d/\tau)$
    Reorder channels: $\mathbf{X}'_d = \mathbf{M}_d^\top \mathbf{X}_d^{\text{batch}}$
  end for
  Compute total loss: $\mathcal{L}_{\text{stage1}} = \mathcal{L}_{\text{corr}} + \lambda_{\text{entropy}} \mathcal{L}_{\text{entropy}}$
  Update parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{stage1}}$
  Update temperature: $\tau \leftarrow \max(\tau_{\text{final}}, \tau_{\text{initial}} \cdot \gamma^{\text{epoch}})$
end for

The two components of the loss function guarantee that the rearranged signal on day k closely resembles that of day 0 while also optimizing the approximation of the permutation matrix. After training, the parameters of the permutation network are fixed for the subsequent training or evaluation phase.

2.2. Reconstruction: Channel-based VQ-VAE

Firstly, we formulate the tasks of channel reconstruction. Then, we describe the architecture of our channel-base VQ-VAE. Next, we introduce the loss function of our reconstruction training.

2.2.1. Problem Formulation

Given neural signal data from day 0 and the rearranged data from day k, which are mixed together as X_{mix} in a ratio of 7:3. Our goal is to reconstruct missing or corrupted segments in the neural signals. During training, we simulate such conditions by randomly masking a signal segment from one of the channels in each sample. Specifically, for each sample n, we randomly select a channel c_n and a time interval $[ti_{\text{start}}, ti_{\text{end}}]$, and set:

$$\tilde{\mathbf{x}}_{mix,n,c_n}[ti_{\text{start}} : ti_{\text{end}}] = 0, \tag{15}$$

where $\tilde{\mathbf{x}}_{mix,n,c_n}$ is the masked signal for sample n and channel c_n.

2.2.2. VQ-VAE Architecture

The architecture of the VQ-VAE is illustrated in Figure 2. It consists of an encoder that independently embeds the

raw time series from different channels as tokens, a cross-channel attention module that captures inter-channel dependencies, and a decoder comprising an MLP network that reconstructs the signals. The VQ-VAE also includes a vector quantization layer for discretizing the latent representations.

Channel Encoder The encoder E_ϕ processes the masked signals to produce latent representations.

For each sample n, channel c and time Ti, the raw signals $\tilde{\mathbf{x}}_{0,n,c} \in \mathbb{R}^{Ti}$ is embedded using a fully connected layer to produce a token representation:

$$\mathbf{e}_{n,c} = \text{Embed}(\tilde{\mathbf{x}}_{mix,n,c}) = \mathbf{W}_e \tilde{\mathbf{x}}_{mix,n,c} + \mathbf{b}_e \in \mathbb{R}^{d_e}, \tag{16}$$

where $\mathbf{W}_e \in \mathbb{R}^{d_e \times T}$ and $\mathbf{b}_e \in \mathbb{R}^{d_e}$ are learnable parameters, and d_e is the embedding dimension.

The embeddings from all channels are input into a cross-channel attention mechanism to capture dependencies among channels. We stack the embeddings for each sample to form $\mathbf{E}_n = [\mathbf{e}_{n,1}; \mathbf{e}_{n,2}; \ldots; \mathbf{e}_{n,C}] \in \mathbb{R}^{C \times d_e}$.

We apply a self-attention mechanism (Vaswani, 2017):

$$\mathbf{H}_n = \text{MultiHeadAttention}(\mathbf{E}_n), \tag{17}$$

where $\mathbf{H}_n \in \mathbb{R}^{C \times d_e}$ contains the attention representations for each channel.

Each attention representation $\mathbf{h}_{n,c} \in \mathbb{R}^{d_e}$ is passed through a feed-forward network to obtain the latent representation:

$$\mathbf{z}_{n,c} = \text{ReLU}\left(\mathbf{h}_{n,c}\mathbf{W}_1 + \mathbf{b}_1\right)\mathbf{W}_2 + \mathbf{b}_2 \in \mathbb{R}^d, \tag{18}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_e \times d_{\text{ff}}}$, $\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$, and $\mathbf{b}_1$, $\mathbf{b}_2$ are biases.

Vector Quantization Layer We use a codebook $\mathcal{C} = \{\mathbf{e}_k\}_{k=1}^K$, where each codeword $\mathbf{e}_k \in \mathbb{R}^d$. Each latent vector $\mathbf{z}_{n,c}$ is quantized by finding the nearest codeword:

$$\mathbf{q}_{n,c} = \mathbf{e}_{k^*}, \quad \text{where } k^* = \arg\min_k \|\mathbf{z}_{n,c} - \mathbf{e}_k\|_2^2. \tag{19}$$

MLP Decoder The decoder D_ψ reconstructs the signals from the quantized embeddings. It consists of a feed-forward network:

$$\hat{\mathbf{x}}_{n,c} = \mathbf{W}_d \mathbf{q}_{n,c} + \mathbf{b}_d \in \mathbb{R}^T, \tag{20}$$

where $\mathbf{W}_d \in \mathbb{R}^{T \times d}$ and $\mathbf{b}_d \in \mathbb{R}^T$ are learnable parameters.

Algorithm 2 Renconstruction Training

Input: Day 0 and rearranged day k data, which are mixed together as $\mathbf{X}_{mix}$ in a ratio of 7:3, masking probability p_{mask}, loss weights λ_{freq}, λ_{vq}, commitment cost β
Initialize: Encoder parameters ϕ, decoder parameters ψ
Initialize codebook $\mathcal{C}$ using k-means clustering
for each epoch **do**
  for each minibatch **do**
    Apply masking to obtain $\tilde{\mathbf{X}}_{mix}^{\text{batch}}$
    Compute embeddings $\mathbf{E}_n$ for each sample
    Perform attention mechanism to obtain $\mathbf{A}_n$
    Obtain latent representations $\{\mathbf{z}_{n,c}\}$
    Quantize latent representations to get $\{\mathbf{q}_{n,c}\}$
    Decode to obtain reconstructed signals $\{\hat{\mathbf{x}}_{n,c}\}$
    Compute loss $\mathcal{L}_{\text{stage2}}$
    Update ϕ, ψ using gradients of $\mathcal{L}_{\text{stage2}}$
    Update codebook $\mathcal{C}$ using exponential moving average
  end for
end for

2.2.3. Loss Function For Reconstruction Training

We train the VQ-VAE by minimizing a loss function, including reconstruction and commitment losses.

Time Domain Reconstruction Loss We compute the mean squared error (MSE) between the reconstructed signals and the original signals for the masked entries $m_{n,c}$:

$$\mathcal{L}_{\text{time}} = \frac{1}{\sum_{n,c} m_{n,c}} \sum_{n,c} m_{n,c} \|\hat{\mathbf{x}}_{n,c} - \mathbf{x}_{n,c}\|_2^2. \tag{21}$$

Frequency Domain Reconstruction Loss We also compute the MSE in the frequency domain features (amplitude $\mathcal{A}(\cdot)$ and phase $\phi(\cdot)$ via Discrete Fourier Transform (DFT)):

$$\mathcal{L}_{\mathcal{A}} = \frac{1}{\sum m_{n,c}} \sum_{n,c} m_{n,c} \left\|\mathcal{A}(\hat{\mathbf{x}}_{n,c}) - \mathcal{A}(\mathbf{x}_{n,c})\right\|_2^2 \tag{22}$$

$$\mathcal{L}_{\phi} = \frac{1}{\sum m_{n,c}} \sum_{n,c} m_{n,c} \left\|\phi(\hat{\mathbf{x}}_{n,c}) - \phi(\mathbf{x}_{n,c})\right\|_2^2 \tag{23}$$

$$\mathcal{L}_{\text{freq}} = \mathcal{L}_{\mathcal{A}} + \mathcal{L}_{\phi} \tag{24}$$

Vector Quantization Commitment Loss The commitment loss encourages the encoder outputs to be close to the codewords:

$$\mathcal{L}_{\text{vq}} = \frac{1}{NC} \sum_{n,c} \|\text{sg}[\mathbf{z}_{n,c}] - \mathbf{q}_{n,c}\|_2^2 + \beta \|\mathbf{z}_{n,c} - \text{sg}[\mathbf{q}_{n,c}]\|_2^2, \tag{25}$$

where $\text{sg}[\cdot]$ denotes the stop-gradient operator, and β is the commitment cost weight.

Total Loss The total loss function is:

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{time}} + \lambda_{\text{freq}}\mathcal{L}_{\text{freq}} + \lambda_{\text{vq}}\mathcal{L}_{\text{vq}}, \tag{26}$$

where λ_{freq} and λ_{vq} are hyperparameters.

Figure 2 illustrates the conceptual CRRL training and evaluation processes. Algorithms 1 and 2 specify the two stages of CRRL training in detail. In the rearrangement training, the first step is to predict the permutation matrices. Then, CRRL rearranges the spike channels and obtains the rearranged spike. In the reconstruction training, we predict the masked signal and its frequency feature. During optimization, our approach learns to robustly restore the information of the day 0 signal. For evaluation, we can obtain the reconstructed data by well-trained CRRL. The obtained data can decode downstream tasks by a decoder trained on day 0.

3. Experiments

3.1. Setup

We evaluate our method for experiments on simulation and real neural datasets. For the simulation dataset, we simulate two channel-wise variability in the simulation dataset: 1) new or lost neurons and 2) drifted neurons. For 1), we delete/add part of the input channels. For 2), we randomly shuffle the input channels. Additionally, we set different change ratios (5 % or 10 %) to compare the impact of different ratios on performance. To simulate the plasticity of neurons, we also replace part of the neurons with some reserved neurons that have different tuning curves. For real datasets, we use two human datasets and five monkey datasets. The human datasets include a handwriting dataset (Willett et al., 2021) and a speech dataset (Willett et al., 2023). And the monkey datasets (Dyer et al., 2017) comprise two Center-out task datasets ((C) and (M)), two isometric wrist task datasets ((J) and (S)), and a key grasping task dataset (G).

To evaluate and compare the performance of models in different periods, we divided all the datasets into five different time intervals according to the number of days from day 0. The dataset and split details are described in Appendix A.

Baseline Methods. We compared our method with five cross-day decoding methods used in BCI. 1) A manifold-based method Stabilizedbci (Degenhart et al., 2020); 2) Another manifold-based method NoMAD (Karpowicz et al., 2022); 3) A domain adaptation method SD-Net (Fang et al., 2023); 4) An adversarial domain adaptation method ADAN (Farshchian et al., 2018); 5) Another adversarial domain adaptation method Cycle-GAN (Ma et al., 2023). The detailed hyper-parameter settings of these methods are described in Appendix B.

We used SVM to perform classification tasks for decoding in downstream tasks and Wiener Filter (Chen et al., 2006) to perform regression tasks.

Table 1. Albation Study on Simulation Dataset.

R^2/acc	w/o RA	w/o RC	RA + RC
New/lost neuron (5 %)	0.83/99.1	0.72/92.7	**0.92/99.5**
New/lost neuron (10 %)	0.81/96.1	0.69/89.3	**0.87/99.3**
Shuffled channel (5 %)	0.15/55.1	0.85/92.1	**0.89/98.0**
Shuffled channel (10 %)	0.05/48.3	0.77/87.5	**0.86/98.2**
Changed function (5 %)	0.66/80.8	0.45/64.7	**0.73/91.4**

3.2. Results on Simulation Dataset

We evaluate CRRL under three simulated neural instabilities (Table 1). We ablate either the Rearrangement Module (RA) or Reconstruction Module (RC) to analysis module contributions. Here, we use Coefficient of Determination (R^2) and accuracy (acc) to evaluate the performance of regression and classification on the simulation dataset, respectively.

Table 1 reported that using two modules together gives the best performance in all scenarios. For the scenario of new/lost neurons, without RC (w/o RC) degrades performance significantly (0.92/99.5% to 0.72/92.7%), indicating RC's critical role in compensating for neuron loss. For the scenario of drifted neurons, w/o RA gives the worst performance, and w/o RC achieves the second-best performance, suggesting RA is vital for handling neuron drifting. When tuning function changes, RA+RC achieves 91.4% accuracy with a 26.7% relative improvement over the w/o RC. RC contributes more to R^2 and acc (0.66/80.8% compared to 0.45/64.7%), implying it effectively adapts to latent representational shifts.

3.3. Results on Speech and Handwriting Decoding

We compare our method with SD-Net and NoMAD on speech and handwriting datasets. The decoding target of the speech dataset is 7 words with a "do nothing" condition, where we use acc as the evaluation metric. Similarly, the decoding target of the handwriting dataset is 31 characters, and we also use acc as the metric.

Figure 3 shows CRRL's improvements in cross-day decoding performance with SD-Net and NoMAD. For the speech dataset, CRRL shows consistent performance on different days. Notably, CRRL achieves accuracy recovery to 86% on day 21 while accuracies of SD-Net and NoMAD decline to 62% and 42% respectively. Moreover, the performance advantage of CRRL is more obvious when the time span is longer. On the last day (day 28), CRRL preserves 71% accuracy (18% total decline), outperforming SD-Net's 65% (25% loss) and NoMAD's 38% (56% loss).

Table 2. The performance of regression and classification prediction which compares with different cross-day decoding methods.

R^2/acc	Model	Task	Number of days since day 0					
			0	[5, 10)	[10, 20)	[20, 40)	[40, 65)	[65, $+\infty$)
Center-Out (C)	Stabilizedbci	Trajectory/ Direction	0.84/86.5	0.59/56.6	0.52/50.7	0.45/44.8	-/-	-/-
	SD-Net			0.73/69.3	0.65/62.5	0.62/63.3	-/-	-/-
	NoMAD			0.55/48.4	0.58/50.6	0.41/37.9	-/-	-/-
	CRRL (Ours)			**0.75/77.1**	**0.68/70.5**	**0.65/66.8**	-/-	-/-
Center-Out (M)	Stabilizedbci	Trajectory/ Direction	0.68/73.1	-/-	0.38/51.6	0.34/43.7	-/-	-/-
	SD-Net			-/-	0.45/62.0	0.39/52.0	-/-	-/-
	NoMAD			-/-	0.33/39.5	0.26/29.7	-/-	-/-
	CRRL (Ours)			-/-	**0.51/63.5**	**0.45/57.4**	-/-	-/-
ISO (J)	Stabilizedbci	Trajectory/ Direction	0.73/92.5	0.38/57.4	0.32/53.1	0.23/35.0	0.20/36.5	0.17/29.1
	SD-Net			0.42/65.2	0.35/58.6	0.28/44.2	0.25/43.8	0.20/34.5
	NoMAD			0.44/65.2	0.45**62.4**	0.37/40.8	0.28/37.5	0.31/39.7
	CRRL (Ours)			**0.51/66.3**	**0.48**/61.5	**0.43/62.2**	**0.46/48.6**	**0.47/51.1**
ISO (S)	Stabilizedbci	Trajectory/ Direction	0.75/94.4	0.33/43.3	0.35/47.2	0.24/40.9	0.18/25.0	$<$0 /14.8
	SD-Net			0.40/56.5	0.37/53.7	0.36/53.7	0.32/45.6	0.24/35.9
	NoMAD			0.34/53.3	**0.40**/51.2	**0.38**/45.0	0.37/44.1	0.29/31.3
	CRRL (Ours)			**0.43/57.2**	0.39/**56.6**	0.35/**54.5**	**0.37/54.1**	**0.30/45.2**
ISO (J)	ADAN	EMG	0.71	0.66	0.60	0.56	0.54	0.48
	Cycle-GAN			0.68	0.65	0.63	0.59	0.50
	CRRL (Ours)			**0.70**	**0.66**	**0.69**	**0.65**	**0.61**
Key (G)	ADAN	EMG	0.46	0.26	0.21	0.18	$<$0	-
	Cycle-GAN			0.27	0.25	0.20	0.09	-
	CRRL (Ours)			**0.41**	**0.37**	**0.35**	**0.32**	-

For the handwriting dataset, Our CRRL framework maintains superior long-term performance, outperforming existing methods significantly. CRRL shows stability with 67% accuracy at Day 16 (vs SD-Net's 57% and NoMAD's 32%). Notably, CRRL rebounds to 69% on Day 24 while other methods continue declining. Over 52 days, CRRL preserves 56% accuracy (21% total decline), contrasting sharply with SD-Net's 30% (58% loss) and NoMAD's 21% (70% loss).

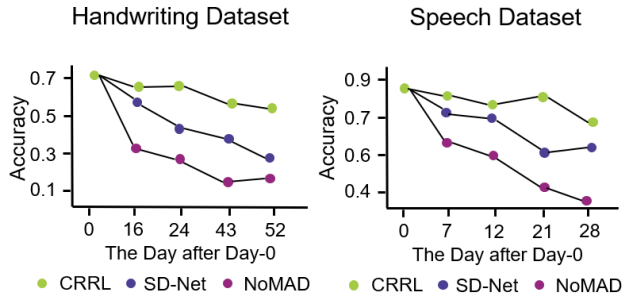

Figure 3. The cross-day decoding performance on handwriting and speech dataset.

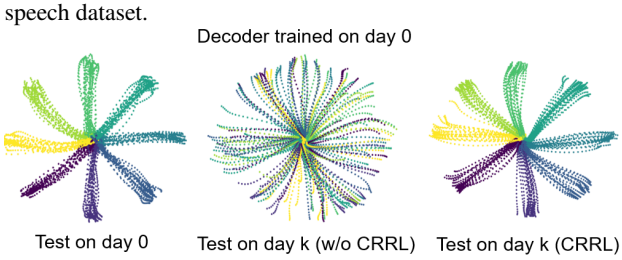

Figure 4. Visualization of the decoding results of hand trajectories, comparing with and without CRRL.

3.4. Results on Motor Decoding

We compare CRRL with Stabilizedbci, SD-Net, and NoMAD on five monkey neural datasets, which include diverse motor decoding tasks. For hand trajectory and electromyography (EMG) decoding, we use $R2$ as an evaluation metric. For the eight-direction classification, we use accuracy as a metric.

Table 2 shows that CRRL achieves consistent superiority in both trajectory regression (R^2) and direction classification (acc) tasks, particularly under challenging long-term decoding scenarios. For the Center-Out datasets (C and M), CRRL maintains stable performance as days progress. On the Center-Out (C), CRRL achieves R^2/acc of 0.75/77.1% for days [5, 10), surpassing SD-Net (0.73/69.3%) and Stabilizedbci (0.59/56.6%). Notably, even after 20–40 days, CRRL retains 0.65/66.8, exhibiting only a 12.8% drop in R^2 compared to SD-Net's 15.1% decline. On the ISO (J) dataset, CRRL achieves 0.47/51.1% R^2/acc at [65, $+\infty$) days, outperforming NoMAD (0.31/39.7%) and SD-Net (0.20/34.5%). This suggests that our framework effectively mitigates neurons or electrode shifts over time. For the ISO (S) dataset, CRRL keeps classification accuracy above 45% even after 65 days, while Stabilizedbci collapses to $<$0/14.8%, demonstrating CRRL's robustness. On EMGs decoding tasks of ISO (J), CRRL achieves 0.61 R^2 at [65, $+\infty$) days, outperforming Cycle-GAN (0.50) and ADAN (0.48). For the challenging Key (G) dataset, CRRL maintains 0.35 R^2 at [20, 40) days, a 94% improvement over Cycle-GAN (0.20).

Table 3. Ablation Study on Real Neural Datasets.

	Component		Center-Out (C)			ISO (J)			Key (G)		
	RA	RC	[5, 10)	[10, 20)	[20, 40)	[5, 10)	[10, 20)	[20, 40)	[5, 10)	[10, 20)	[20, 40)
CRRL	✓	-	0.73/68.1	0.61/64.6	0.57/63.5	0.44/57.7	0.41/48.3	0.41/52.5	0.35	0.31	0.32
CRRL	-	✓	0.73/66.4	0.64/57.6	0.52/53.2	0.47/62.2	0.36/53.0	0.32/49.3	0.38	0.33	0.26
CRRL	✓	✓	**0.75/77.1**	**0.68/70.5**	**0.65/66.8**	**0.51/66.3**	**0.48/61.5**	**0.43/62.2**	**0.41**	**0.37**	**0.35**

Table 4. Plugin-in experiment.

R^2	Day 7	Day 30	Day 56
ADAN	0.26	0.18	<0
Cycle-GAN	0.27	0.20	0.09
RA + ADAN	0.29	0.23	0.14
RA + Cycle-GAN	**0.33**	**0.27**	**0.20**

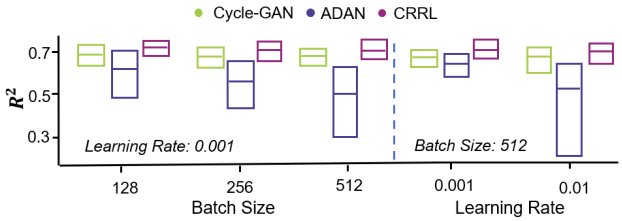

Figure 5. Parameter sensitivity.

3.5. Ablation Study

We conduct ablation study experiments on Center-Out (C), ISO (J) and Key (G) datasets to investigate the effectiveness of two modules of CRRL (shown in Table 3). The decoding target of ISO (J) is hand trajectory. On the Center-Out (C) dataset, RA + RC achieves 0.75 R^2 in [5,10) days, which is 2.7% higher than RA-alone (0.73) and RC-alone (0.73). This advantage of performance grew over time, delivering 12.3% accuracy improvement (66.8% vs 63.5%/53.2%) at [20,40) days. Results on the ISO (J) dataset demonstrate that RA + RC achieves classification accuracy of 62.2% at [20,40) days, outperforming individual components by 18.7% (RC) and 18.4% (RA). Notably, RC proves particularly crucial for Key (G) EMG decoding, where RA achieves 0.38 accuracy at [5,10) days vs RA's 0.35. However, RA + RC yields 7.9% additional gain (0.41).

3.6. Combining CRRL module into DA methods

Since our RC module can be regarded as a special domain adaptation method, and RA is a pre-alignment before adaptation, it can be incorporated with the existing DA method as a plug-in. We evaluate the performance of plug-ins on the Key (G) dataset. As demonstrated in Table 4. When combined with Cycle-GAN, the RA module boosts R^2 scores by 22.2% on Day 7 (0.33 vs 0.27 baseline) and maintains 122% higher accuracy by Day 56 (0.20 vs 0.09). Similar improvements are shown in ADAN. Notably, the RA+Cycle-GAN achieves better performance separation over time: 6.7% advantage on Day 7 expands to 11.1% by Day 30 and 11.0% on Day 56. These results suggest CRRL's effectiveness design enables plug-and-play enhancement of DA methods.

3.7. Analysis of Parameter Sensitivity

We compare the parameter sensitivity of our method with Cycle-GAN and ADAN. In the experiment, we selected a recording in the ISO (J) dataset (Day 30), and the decoding target was EMG. We compare the performance of the models with different batch sizes (BS) and learning rates (LR), respectively. Here LR is only set to the LR of the decoder. When comparing one of the parameters, the other parameter is fixed, as shown in the bottom left of Figure 5. Results show that ADAN can fluctuate greatly and even degrade the performance significantly under different parameter Settings. However, our CRRL and Cycle-GAN are more stable. In contrast, our method performs better than Cycle-GAN in all settings.

4. Limitation and Conclusion

In this work, we propose CRRL, a Channel Rearrangement and Reconstruction Learning framework. Our model is inspired by the channel-wise variability in neural signals. By adapting different types of variability in the model, it effectively alleviates the damage of this variability on the decoder and achieves state-of-the-art performance on multiple tasks. CRRL has good robustness, which makes its advantage more obvious when the time span is long. In addition, CRRL can also be combined with other methods as plug-ins to improve the performance of other methods.

However, although our work achieves better performance than the existing methods, we still observe a slow decline in performance over time, which may be due to slow changes in neuronal population modulation caused by learning or environmental changes. How to accurately capture the modulation change patterns of neuronal populations is another interesting direction for future work. Overall, we hope the proposed cross-day decoding framework will inspire new intelligence paradigms and provide a tool for long-term stable clinical BCIs.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ajiboye, A. B., Willett, F. R., Young, D. R., Memberg, W. D., Murphy, B. A., Miller, J. P., Walter, B. L., Sweet, J. A., Hoyen, H. A., Keith, M. W., et al. Restoration of reaching and grasping movements through brain-controlled muscle stimulation in a person with tetraplegia: a proof-of-concept demonstration. *The Lancet*, 389 (10081):1821–1830, 2017.

Azabou, M., Arora, V., Ganesh, V., Mao, X., Nachimuthu, S., Mendelson, M., Richards, B., Perich, M., Lajoie, G., and Dyer, E. A unified, scalable framework for neural population decoding. *Advances in Neural Information Processing Systems*, 36, 2024.

Card, N. S., Wairagkar, M., Iacobacci, C., Hou, X., Singer-Clark, T., Willett, F. R., Kunz, E. M., Fan, C., Vahdati Nia, M., Deo, D. R., et al. An accurate and rapidly calibrating speech neuroprosthesis. *New England Journal of Medicine*, 391(7):609–618, 2024.

Chen, J., Benesty, J., Huang, Y., and Doclo, S. New insights into the noise reduction wiener filter. *IEEE Transactions on audio, speech, and language processing*, 14(4):1218–1234, 2006.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. *NeurIPS*, 2013.

Degenhart, A. D., Bishop, W. E., Oby, E. R., Tyler-Kabara, E. C., Chase, S. M., Batista, A. P., and Yu, B. M. Stabilization of a brain–computer interface via the alignment of low-dimensional spaces of neural activity. *Nature biomedical engineering*, 4(7):672–685, 2020.

Dyer, E. L., Gheshlaghi Azar, M., Perich, M. G., Fernandes, H. L., Naufel, S., Miller, L. E., and Körding, K. P. A cryptography-based approach for movement decoding. *Nature biomedical engineering*, 1(12):967–976, 2017.

Fang, T., Zheng, Q., Qi, Y., and Pan, G. Extracting semantic-dynamic features for long-term stable brain computer interface. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 5965–5973, 2023.

Farshchian, A., Gallego, J. A., Cohen, J. P., Bengio, Y., Miller, L. E., and Solla, S. A. Adversarial domain adaptation for stable brain-machine interfaces. *arXiv preprint arXiv:1810.00045*, 2018.

Gallego, J. A., Perich, M. G., Miller, L. E., and Solla, S. A. Neural manifolds for the control of movement. *Neuron*, 94(5):978–984, 2017.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *ICLR*, 2017a.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations*, 2017b.

Karpowicz, B. M., Ali, Y. H., Wimalasena, L. N., Sedler, A. R., Keshtkaran, M. R., Bodkin, K., Ma, X., Miller, L. E., and Pandarinath, C. Stabilizing brain-computer interfaces through alignment of latent dynamics. *BioRxiv*, pp. 2022–04, 2022.

Kuhn, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955.

Ma, X., Rizzoglio, F., Bodkin, K. L., Perreault, E., Miller, L. E., and Kennedy, A. Using adversarial networks to extend brain computer interface decoding accuracy over time. *elife*, 12:e84296, 2023.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Metzger, S. L., Littlejohn, K. T., Silva, A. B., Moses, D. A., Seaton, M. P., Wang, R., Dougherty, M. E., Liu, J. R., Wu, P., Berger, M. A., et al. A high-performance neuroprosthesis for speech decoding and avatar control. *Nature*, 620(7976):1037–1046, 2023.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Vaswani, A. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Willett, F. R., Avansino, D. T., Hochberg, L. R., Henderson, J. M., and Shenoy, K. V. High-performance brain-to-text communication via handwriting. *Nature*, 593(7858): 249–254, 2021.

Willett, F. R., Kunz, E. M., Fan, C., Avansino, D. T., Wilson, G. H., Choi, E. Y., Kamdar, F., Glasser, M. F., Hochberg, L. R., Druckmann, S., et al. A high-performance speech neuroprosthesis. *Nature*, 620(7976):1031–1036, 2023.

Ye, J., Collinger, J., Wehbe, L., and Gaunt, R. Neural data transformer 2: multi-context pretraining for neural spiking activity. *Advances in Neural Information Processing Systems*, 36, 2024.

A. Experiments Details

A.1. Dataset Details

In Section 3, we have described the datasets used in the experiments (Dyer et al., 2017; Willett et al., 2023; 2021). Here, we give their details and download links:

- **Isometric wrist (ISO)** .[1] In the Isometric wrist datasets, Monkeys J and S performed an isometric wrist task, controlling a cursor by applying forces on a padded box. Each trial began with a center target hold (0.2–1.0 s), followed by a randomly selected outer target and an auditory cue. EMG data were collected from 7 muscles in Monkey J's left arm (ECU, FCU, ECRl, EDC, FCR, FDP, ECRb) and 8 muscles in Monkey S's left arm and hand (FDP1, FDP2, FCR, 1DI, FPB, EDC, MD, ECRl). Trial lengths varied as no temporal alignment was performed.
- **Center-out reach**.[1] In the Center-Out datasets, Monkeys C and M performed a center-out reaching task using a planar manipulandum, moving the handle in a parasagittal plane. Each trial began with the monkey moving to the center of the workspace, followed by a random waiting period before one of eight outer targets appeared. Trials were extracted from the 'gocue time' to the 'trial end', as the monkeys remained stationary before the gocue.
- **Key Grasping**.[1] In Key Grasping dataset, Monkey G performed a grasping task, reaching and grasping a small rectangular cuboid under the screen using a key grasp with the thumb and index finger. Each trial began with the monkey resting its hand on a touchpad for a random period (0.5–1.0 s). Trials were extracted from the 'gocue time' to the 'trial end', as movements were random before the gocue. EMG data were collected from 8 muscles in the left arm and hand (FCR, FDP, PT, FPB, 1DI, SUP, ECU, EDC).
- **Handwriting**.[2] In the handwritten dataset, the neural activity was recorded from two microelectrode arrays which including 192 channels in the motor area. In the experiments of handwriting dataset, the participant attempted to write 31 different characters in response to cues shown on a computer screen. The cue list includes the following characters:'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'greaterThan', 'comma', 'apostrophe', 'tilde', 'questionMark'.
- **Speech**.[3] In the speech dataset, the neural activity was recorded from four microelectrode arrays (each array includes 64 channels and we only use the first two arrays here). In the experiments of speech dataset, the participant attempted to speak 7 single words (and a "do nothing" condition) in response to cues shown on a computer. The cue list includes the following words: 'choice', 'day', 'kite', 'though', 'veto', and 'were'.

More parameters of these datasets are shown in Table 5.

Table 5. Detailed dataset parameters.

Dataset	Range of days	Num of recordings	Day 0	Decoding target	Decoding Task
ISO (J)	95	20	20150730	EMGs, hand trajectory and 8 directions	Regression & Classification
ISO (S)	83	18	20120821	EMGs, hand trajectory and 8 directions	Regression & Classification
Center-Out (C)	38	12	20160927	Hand trajectory and 8 directions	Regression & Classification
Center-Out (M)	32	11	20140203	Hand trajectory and 8 directions	Regression & Classification
Key (G)	53	8	20190812	EMGs	Regression
Handwriting	51	10	20191125	31 characters	Classification
Speech	41	9	20220616	7 words and a "do nothing" condition	Classification

A.2. Dataset Split

To investigate the decoder's performance changes over different time periods, we divide all the datasets into five time periods: [5, 10), [10, 20), [20, 40), [40, 65), [65, $+\infty$). The data before day 5 includes day 0 as training data (Figure 6). In particular,

[1] https://datadryad.org/stash/dataset/doi:10.5061/dryad.cvdncjt7n
[2] https://datadryad.org/stash/dataset/doi:10.5061/dryad.wh70rxwmv
[3] https://datadryad.org/stash/dataset/doi:10.5061/dryad.x69p8czpq

in the handwriting and Center-Out (M) datasets, there is no data before day 5, and we take the training set to the first day after day 0. For each time period, we compute the average of all data performance in the time period. We will average performance across 10 random seeds for each data point.

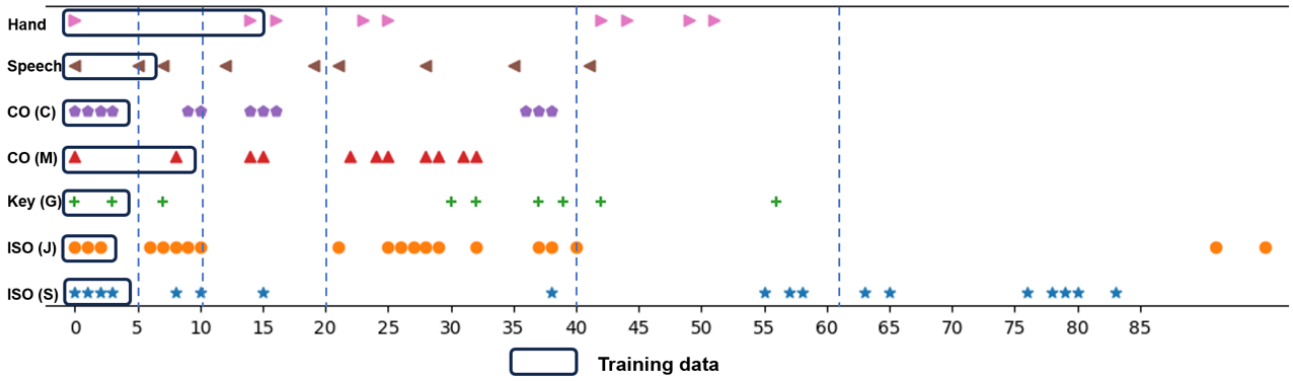

Figure 6. Visualization of the decoding results, comparing with and without CRRL. (a) The decoding result of hand trajectories. (b) The decoding result of EMG.

A.3. Metric Details

We adopt the **Accuracy (ACC)** metric to evaluate classification tasks (8 directions, phonemes and characters), which measures the proportion of correctly predicted samples. For regression tasks (hand trajectory), we use the **Coefficient of Determination (R^2)** to assess the proportion of variance in the target variable explained by the model. In multivariate regression tasks (EMGs), we extend R^2 to **Multi-Target R^2** to evaluate the model's performance across multiple output variables simultaneously. These metrics are calculated as follows:

Accuracy (ACC):

$$\text{ACC} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(y_i = \hat{y}_i) \tag{27}$$

where y_i represents the true label, $\hat{y}_i$ represents the predicted label, $\mathbb{I}(\cdot)$ is the indicator function, and n is the total number of samples.

Coefficient of Determination (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{28}$$

where y_i represents the actual value, $\hat{y}_i$ represents the predicted value, and $\bar{y}$ is the mean of the actual values.

Multi-Variate R^2:

$$\text{Multi-Variate } R^2 = 1 - \frac{\sum_{j=1}^{m}\sum_{i=1}^{n}(y_{i,j} - \hat{y}_{i,j})^2}{\sum_{j=1}^{m}\sum_{i=1}^{n}(y_{i,j} - \bar{y}_j)^2} \tag{29}$$

where $y_{i,j}$ represents the actual value of the j-th target for the i-th sample, $\hat{y}_{i,j}$ represents the predicted value of the j-th target for the i-th sample, $\bar{y}_j$ is the mean of the actual values for the j-th target, m is the number of targets, and n is the number of samples.

These metrics collectively offer a comprehensive evaluation of model performance across diverse tasks.

B. Hyperparameters Settings

B.1. Our CRRL

We undertake several practical considerations to ensure adequate training and inference in our implementation. We adopt the Adam optimizer with a learning rate of 0.001 for model training and utilize a batch size of 32 samples. The hyperparameters, such as the weights of different loss λ_{freq} and λ_{vq} are chosen based on validation performance. During masked channel

Table 6. Hyperparameters for rearrangement training.

Hyperparameters	Values
Batch Size	128
Initial Temperature τ_{initial}	1
Shuffling Rate	0.1
Final Temperature τ_{final}	0.001
Decay Rate γ	0.01
Entropy Coefficient λ_{entropy}	0.2
Number of Training Epochs	300
Learning Rate	0.001

Table 7. Hyperparameters for reconstruction training.

Hyperparameters	Values
Batch Size	256
Masking Ratio	0.05
Embedding Size of Encoder	128
Num of Attention Head	4
Codebook Size	1024, 64
Commitment Cost Weight β	0.25
λ_{freq}	0.1
λ_{vq}	0.2
Number of Training Epochs	500
Learning Rate	0.001

modeling, a masking ratio of 0.05 determines the proportion of channels to mask in each input, and a random channel shuffling rate of 0.1 is applied during the training of the permutation network to enhance robustness.

The embedding size of encoder E_θ is 128, and the number of attention heads is 4. The codebook size K is 512, and the commitment cost weight β is set to 0.25 to balance expressiveness and computational efficiency. Our model is implemented using PyTorch, enabling flexible model design and training, and training and inference are performed on GPUs to accelerate computation, with specific experiments conducted on NVIDIA Tesla A100 GPUs.

B.2. Stabilizedbci

Stabilizedbci (Degenhart et al., 2020) aims to stabilize brain-computer interfaces (BCIs) by leveraging the concept of a "neural manifold," a low-dimensional representation of population-level neural activity. The proposed approach addresses the challenge of neural recording instabilities, such as electrode shifts or neuron dropout, which can disrupt BCI performance. By aligning the neural manifold across different time points using stable electrodes as reference points, the method maintains a consistent mapping of neural activity to movement intent. This eliminates the need for frequent recalibration, allowing the BCI to provide reliable performance even under severe recording instabilities, and it does so without requiring knowledge of the user's intent.

Table 8. Hyperparameters for Stabilizedbci training.

Hyperparameters	Values
Dim of Latent	10
Num of FA models	5
Maximum Num of EM iterations	100000
EM stopping criteria	0.00001
Minimum private variance threshold	0.1
Num of rows of loading matrix	90
Alignment L2 norm threshold	0.01

B.3. SD-Net

SD-Net (Fang et al., 2023) focuses on extracting both semantic and dynamic features from neural signals, leveraging the idea that low-dimensional dynamics can represent high-dimensional neural activity. By embedding these features into a unified space, the method enables recalibration without requiring labeled data from new sessions. Additionally, the paper introduces a joint distribution alignment strategy, which aligns both marginal and conditional distributions to handle complex domain shifts in neural data. This approach is validated on real and simulated datasets, demonstrating its effectiveness in improving BCI performance across sessions.

Table 9. Hyperparameters for SD-Net training.

Hyperparameters	Values
Learning Rate	0.002
Batch Size	10
Dim of Latent	256
Epoch	60
λ_1, λ_2	0.1, 0.1
λ_3, λ_4	0.1, 0.1
λ_5, λ_6	0.1, 1

B.4. NoMAD

NoMAD (Nonlinear Manifold Alignment with Dynamics) (Karpowicz et al., 2022) is a novel method to stabilize brain-computer interface (BCI) decoding performance over time. The approach leverages the low-dimensional manifold structure and dynamics of neural activity, using unsupervised learning to align neural data across different time periods. By incorporating recurrent neural network (RNN) models to capture neural dynamics and an alignment network to map neural data to a consistent manifold, NoMAD allows the initial decoder to maintain high accuracy without requiring additional labeled data or frequent recalibration. This method was validated on monkey motor task datasets, demonstrating superior decoding stability and performance compared to existing methods, paving the way for more practical and robust BCI systems

Table 10. Hyperparameters for NoMAD training.

Hyperparameters	Values
Initial learning rate	10
Batch size	5
NLL cost weight	100000
NLL ramping epochs	0.00001
KL ramping epochs	0.1
KL weight on initial conditions	90
KL weight on controller outputs	0.01

Table 11. Hyperparameters for Cycle-GAN training.

Hyperparameters	Values
Batch size	256
Discriminator (D1) learning rate	0.01
Discriminator (D2) learning rate	0.01
Generator (G1) learning rate	0.001
Generator (G2) learning rate	0.001
Number of training epochs	200

B.5. Cycle-GAN

Cycle-GAN (Ma et al., 2023) uses Cycle-Consistent Adversarial Networks to align neural activity recorded on different days, enabling a fixed decoder trained on an initial day to maintain accurate predictions without requiring frequent recalibration. Unlike previous methods, Cycle-GAN operates directly on the full-dimensional neural signals, avoiding information loss

from dimensionality reduction and offering greater robustness and ease of training. The approach outperforms prior alignment techniques like ADAN and Procrustes-based methods, making it a promising solution for long-term iBCI applications.

B.6. ADAN

ADAN (Farshchian et al., 2018) combines a deep learning-based neural autoencoder with an EMG predictor to extract a low-dimensional representation of neural signals while simultaneously predicting movement intent. To stabilize the BMI over time, they implement domain adaptation techniques, including Canonical Correlation Analysis (CCA), Kullback-Leibler Divergence Minimization (KLDM), and a new Adversarial Domain Adaptation Network (ADAN). ADAN, which aligns the probability distributions of residuals from neural signal reconstructions, outperforms other methods and requires minimal data to adapt, offering a more robust and consistent interface for users.

Table 12. Hyperparameters for ADAN training.

Hyperparameters	Values
Dim of Decoder Latent	10
Batch Size of Decoder	64
Epochs of Decoder	400
Learning Rate of Decoder	0.0001
Decoder layers	1
Epochs of ADAN	200
Batch Size of ADAN	4
Learning Rate of Discriminator	0.00001
Learning Rate of Generator	0.0001

C. Gradient Propagation Analysis for Gumbel-Sinkhorn Networks

C.1. Problem Formulation

Given a log-probability matrix $\mathbf{L} \in \mathbb{R}^{C\times C}$ which obtained by our channel permutation network, we seek to solve the combinatorial optimization problem:

$$\mathbf{P} = \arg\max_{\mathbf{P}\in\mathcal{P}_C} \langle \mathbf{L}, \mathbf{P} \rangle, \tag{30}$$

where $\mathcal{P}_C$ denotes the set of $C \times C$ permutation matrices. Direct differentiation through the discrete $\arg\max$ operator is infeasible due to its non-smooth nature.

C.2. Differentiable Reparameterization with Gumbel Noise

Following (Jang et al., 2017b), we inject Gumbel-distributed noise $\epsilon_{i,j} \sim \text{Gumbel}(0, 1)$ into the logits:

$$\tilde{\mathbf{L}}_{i,j} = \mathbf{L}_{i,j} + \gamma\epsilon_{i,j}, \tag{31}$$

where $\gamma > 0$ controls the noise magnitude. This reparameterization preserves the original distributional properties while enabling gradient flow.

C.3. Continuous Relaxation via Entropy-Regularized Optimal Transport

The Gumbel-perturbed logits are transformed into a doubly stochastic matrix $\mathbf{M}$ through the Sinkhorn-Knopp algorithm (Cuturi, 2013). Let $\tau > 0$ be a temperature parameter:

$$\mathbf{K}^{(0)} = \exp\left(\frac{\tilde{\mathbf{L}}}{\tau}\right). \tag{32}$$

Alternating row and column normalizations are applied for T iterations:

$$\mathbf{R}^{(t)} = \text{diag}\left(\mathbf{K}^{(t-1)}\mathbf{1}_N\right)^{-1}, \tag{33}$$

$$\mathbf{K}^{(t)} = \mathbf{R}^{(t)}\mathbf{K}^{(t-1)}, \tag{34}$$

$$\mathbf{C}^{(t)} = \text{diag}\left(\mathbf{1}_N^\top\mathbf{K}^{(t)}\right)^{-1}, \tag{35}$$

$$\mathbf{K}^{(t)} = \mathbf{K}^{(t)}\mathbf{C}^{(t)}, \tag{36}$$

where $\mathbf{1}_N$ is a column vector of ones. The final doubly stochastic matrix is $\mathbf{M} = \mathbf{K}^{(T)}$.

C.4. Differentiability of the Sinkhorn Operator

Proposition C.1 (Implicit Gradient Computation)**.** *The Jacobian $\nabla_{\tilde{\mathbf{L}}}\mathbf{M}$ can be computed via implicit differentiation of the Sinkhorn iterations (Cuturi, 2013; Mena et al., 2018).*

Proof. Let $\mathcal{S}(\tilde{\mathbf{L}}/\tau)$ denote the Sinkhorn operator. The gradient chain through T iterations is:

$$\frac{\partial \mathbf{M}}{\partial \tilde{\mathbf{L}}} = \prod_{t=1}^{T} \frac{\partial \mathbf{K}^{(t)}}{\partial \mathbf{K}^{(t-1)}} \cdot \frac{\partial \mathbf{K}^{(0)}}{\partial \tilde{\mathbf{L}}}. \tag{37}$$

Each term $\frac{\partial \mathbf{K}^{(t)}}{\partial \mathbf{K}^{(t-1)}}$ is derived from the row/column scaling operations, which are element-wise differentiable. Full derivations are provided in (Mena et al., 2018). □

C.5. Convergence to Exact Permutations

Proposition C.2 (Discrete Limit)**.** *As $\tau \to 0^+$, $\mathbf{M}$ converges almost surely to the permutation matrix $\mathbf{P}$ obtained by the Hungarian algorithm on $\tilde{\mathbf{L}}$.*

Proof. When $\tau \to 0^+$, the entries of $\mathbf{K}^{(0)}$ satisfy:

$$\lim_{\tau \to 0^+} \exp\left(\frac{\tilde{\mathbf{L}}_{i,j}}{\tau}\right) = \begin{cases} 1 & \text{if } (i,j) = \arg\max_{k,l} \tilde{\mathbf{L}}_{k,l} \\ 0 & \text{otherwise} \end{cases}. \tag{38}$$

The Sinkhorn iterations preserve the dominant permutation pattern, recovering the Hungarian solution (Mena et al., 2018). □