# NeuroSim: A Gymnasium-Compatible Reinforcement Learning Environment Suite for Brain-Computer Interfaces

Hass Dhia
Smart Technology Investments Research Institute
Los Angeles, CA, USA
partners@smarttechinvest.com

February 24, 2026

**Abstract**

Brain-computer interface (BCI) research is constrained by the high cost of human neural recordings, the non-stationarity of neural signals across sessions, and the absence of standardized reinforcement learning (RL) benchmarks. We present **NeuroSim**, an open-source Python library that provides three Gymnasium-compatible RL environments modeling distinct BCI paradigms: `DecoderAdapt-v0` for motor imagery decoder adaptation, `CursorControl-v0` for continuous intracortical cursor control, and `SpellerNav-v0` for P300-based speller stimulus selection. Each environment incorporates physiologically grounded signal models for electrode impedance drift, neural fatigue, feature distribution shifts, and user–decoder co-adaptation. NeuroSim integrates with the MOABB dataset ecosystem for calibration from real EEG data and includes a conditional variational autoencoder (cVAE) surrogate for generating unlimited synthetic training epochs. We report baseline results with Proximal Policy Optimization (PPO) trained for 20,000 timesteps across all three environments. PPO achieves a mean reward of 121.11 on DecoderAdapt-v0 (vs. 23.70 for a random agent) and $-89.99$ on CursorControl-v0 (vs. $-209.21$), while underperforming random on SpellerNav-v0 ($-11.60$ vs. $-5.55$) due to insufficient exploration at this training budget. The platform also provides a classical CSP+LDA baseline for comparison with RL agents and a leave-one-subject-out cross-subject transfer evaluation protocol. The full platform, including 158 passing tests and benchmark infrastructure, is released under the MIT license at `https://github.com/HassDhia/neurosim`.

## 1 Introduction

Brain-computer interfaces translate neural activity into control signals for external devices, enabling communication and motor restoration for individuals with severe neurological conditions. Despite substantial progress in neural decoding algorithms, BCI systems remain fragile in practice: electrode impedance drifts over hours, neural signal statistics shift across sessions, and users co-adapt their neural strategies to the decoder's expectations. These non-stationarities degrade decoder performance and necessitate frequent recalibration, reducing the usability of BCI systems in real-world settings.

Reinforcement learning offers a principled framework for addressing these challenges. An RL agent can learn adaptive policies that respond to signal degradation, balance exploration against exploitation in stimulus selection, and manage the cost–benefit tradeoff of decoder recalibration. However, training RL agents on real neural data is prohibitively expensive and ethically constrained. Each experimental session requires a human participant, specialized recording equipment, and hours of setup time. Simulated environments offer an alternative, but existing BCI simulation tools lack the standardized interfaces, signal fidelity, and benchmark infrastructure needed for rigorous RL research.

This paper introduces **NeuroSim**, a Gymnasium-compatible environment suite designed to bridge this gap. NeuroSim makes three contributions:

1. **Three standardized RL environments** covering the primary BCI paradigms—motor imagery classification, continuous cursor control, and event-related potential (ERP) speller navigation—each with carefully designed observation, action, and reward spaces.

2. **Physiologically grounded signal models** for four sources of non-stationarity (electrode impedance drift, neural fatigue, feature distribution shift, and user–decoder co-adaptation), composable through a modular signal pipeline.

3. **A conditional VAE surrogate** trained on real or synthetic neural epochs that provides unlimited class-conditioned data generation for scalable RL training.

4. **Classical and cross-subject baselines**: a CSP+LDA classifier for comparison with RL agents, and a leave-one-subject-out (LOSO) transfer evaluation protocol for assessing generalization across synthetic subject cohorts.

NeuroSim integrates with the MOABB dataset ecosystem [Jayaram and Barachant, 2018] for calibration from real recordings and provides a benchmark runner with PPO baselines for reproducible comparisons. The entire platform is released as an installable Python package under the MIT license with 158 passing tests, type annotations, and comprehensive documentation.

The remainder of this paper is organized as follows. Section 2 reviews related work in BCI simulation and RL for neural decoding. Section 3 describes the system architecture. Section 4 details the three Gymnasium environments. Section 5 formalizes the signal models. Section 6 presents the cVAE surrogate architecture. Section 7 reports baseline PPO results. Section 8 discusses design tradeoffs and limitations. Section 9 concludes with a summary and future directions.

## 2 Related Work

### 2.1 BCI Simulators

Simulation has long been recognized as essential for BCI algorithm development, but few platforms provide the combination of physiological fidelity and RL-ready interfaces. Liang and Kao [2024] provided a comprehensive review of BCI simulator architectures, identifying key gaps in signal non-stationarity modeling and standardized benchmarking. Their work highlighted the need for modular simulation components that can be composed to model different BCI paradigms and degradation scenarios.

Existing tools for BCI data access include MOABB [Jayaram and Barachant, 2018], which standardizes access to 36+ BCI datasets through a unified Python API covering motor imagery, P300, and steady-state visual evoked potential (SSVEP) paradigms. BrainFlow provides a hardware abstraction layer for real-time neural data acquisition but does not include simulation capabilities. NeuroGym offers Gymnasium-compatible neuroscience environments, but focuses on cognitive neuroscience tasks rather than BCI-specific signal processing challenges.

NeuroSim differentiates itself by combining MOABB's dataset ecosystem with physiologically grounded non-stationarity models and RL-native environment interfaces. Rather than replacing existing tools, NeuroSim builds on them: MOABB data feeds the signal models, which in turn drive the Gymnasium environments.

### 2.2 Reinforcement Learning for BCI

The application of RL to BCI control has been explored from multiple angles. Kao et al. [2025] demonstrated that RL agents can learn to co-adapt with neural decoders, improving closed-loop performance beyond what fixed decoding algorithms achieve. Their work established the theoretical framework for treating BCI control as a partially observable Markov decision process.

Sanchez et al. [2009] initiated the study of reinforcement learning architectures for brain-machine interfaces, showing that RL agents can learn co-adaptive decoder strategies from reward signals alone. Their work on exploiting co-adaptation for symbiotic neuroprosthetic assistants demonstrated that the user and decoder can be modeled as a coupled dynamical system where both parties adapt simultaneously.

The Co-adaptive Policy Actor-Critic (CopAC) framework [Zhang et al., 2024] formalized the co-adaptation problem as a two-player cooperative game between the user and the decoder, each modeled as an RL agent. This perspective directly informs NeuroSim's co-adaptation model, which simulates the user's side of this interaction.

Despite these advances, RL for BCI research has been hampered by the lack of standardized benchmarks. Each study uses different datasets, signal processing pipelines, and evaluation metrics, making cross-study comparisons difficult. NeuroSim addresses this gap by providing a common evaluation framework with fixed environment specifications and baseline agent comparisons.

### 2.3 Gymnasium Ecosystem

The Gymnasium API [Brockman et al., 2016] (originally OpenAI Gym) has become the de facto standard for RL environment interfaces. Its `reset()`/`step()` protocol, coupled with standardized observation and action spaces, enables algorithm-agnostic benchmarking. NeuroSim adopts this interface wholesale, ensuring compatibility with popular RL libraries such as Stable Baselines3 [Raffin et al., 2021], RLlib, and CleanRL. Each NeuroSim environment registers under the `neurosim/` namespace and can be instantiated via `gymnasium.make()`.

## 3 System Architecture

NeuroSim is organized as a layered Python package with clear separation between data ingestion, signal modeling, environment simulation, and agent training. Figure 1 illustrates the data flow through the system.

### 3.1 Data Pipeline

The data pipeline bridges the MOABB ecosystem into NeuroSim's internal representation. The `MOABBLoader` class wraps MOABB's dataset and paradigm API, converting raw EEG recordings into a flat list of `NeuralEpoch` objects—a standardized dataclass containing signal arrays of shape $(C \times T)$ (channels by timepoints), integer class labels, sampling frequency, channel names, and subject/session identifiers.

MOABB and MNE are specified as optional dependencies (`pip install neurosim[data]`), so the core simulation functionality operates without them. When MOABB is unavailable, environments generate synthetic neural signals using built-in noise models. The loader supports eight priority datasets spanning the major BCI paradigms:

- BNCI2014_001: 9 subjects, 2 sessions, 4-class motor imagery
- BNCI2014_004: 9 subjects, 5 sessions, 2-class motor imagery
- BNCI2015_001: 12 subjects, 2–3 sessions, 2-class motor imagery

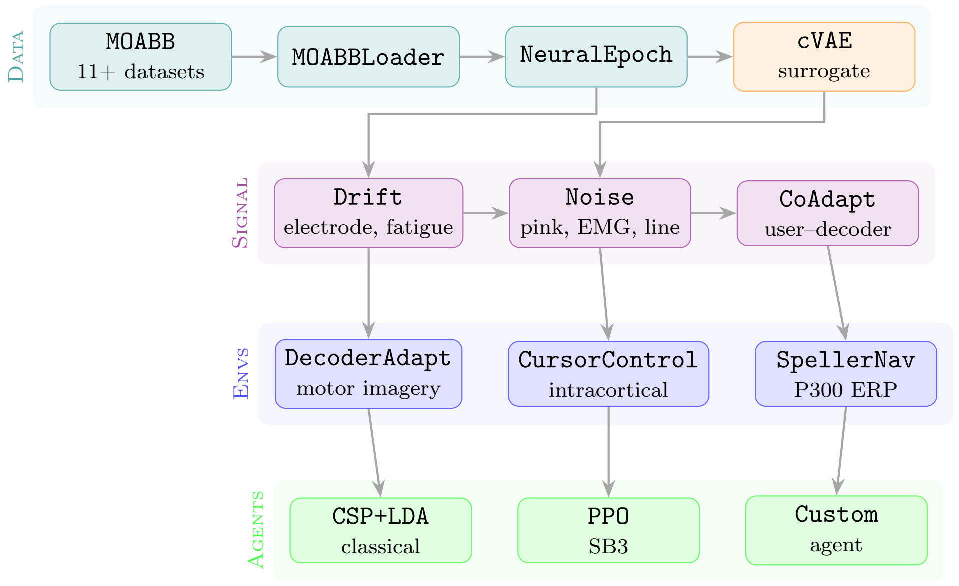

Figure 1: NeuroSim system architecture. Data flows top-to-bottom through four layers: real EEG from MOABB is converted to `NeuralEpoch` objects (with optional cVAE augmentation), processed through a composable signal pipeline of drift, noise, and co-adaptation models, served to three Gymnasium environments, and acted upon by RL or classical agents via the standard `step()` interface.

- Cho2017: 52 subjects, 1 session, 2-class motor imagery
- Lee2019_MI: 54 subjects, 2 sessions, 2-class motor imagery
- PhysionetMI: 109 subjects, 1 session, 4-class motor imagery
- Weibo2014: 10 subjects, 1 session, 7-class motor imagery
- Zhou2016: 4 subjects, 3 sessions, 3-class motor imagery

A preprocessing module provides standard EEG preprocessing operations—bandpass filtering, common average referencing (CAR), z-score normalization, and artifact rejection—composable into a `preprocess_pipeline` function.

Feature extraction converts `NeuralEpoch` objects into environment observation vectors. The `extract_band_power` function computes power spectral density via Welch's method across five standard EEG frequency bands (delta: 0.5–4 Hz, theta: 4–8 Hz, alpha: 8–13 Hz, beta: 13–30 Hz, gamma: 30–45 Hz). The `extract_log_variance` function computes per-channel log-variance, a robust feature inspired by Common Spatial Patterns (CSP).

## 3.2 Environment Design Philosophy

NeuroSim environments follow three design principles:

**Plug-in signal models.** Each environment accepts an optional `signal_pipeline` parameter of type `SignalPipeline`. When `None` (the default), the environment uses built-in inline noise generation. This ensures that the default behavior is unchanged when no pipeline is provided, while allowing researchers to inject arbitrary combinations of drift, noise, and co-adaptation models.

**Dict observation and action spaces.** Environments use Gymnasium's `Dict` spaces to expose semantically meaningful observation components (e.g., `neural_features`, `decoder_confidence`, `cursor_position`) and structured actions (e.g., simultaneous classification and recalibration decisions). For compatibility with RL libraries that require flat spaces, NeuroSim provides a `FlattenDictActionWrapper` and an `ObservationFlattener` utility.

**Gymnasium registration.** All environments register under the `neurosim/` namespace and can be instantiated via `gymnasium.make("neurosim/DecoderAdapt-v0")`. This enables seamless integration with standard RL training loops.

### 3.3 Dependency Architecture

NeuroSim uses a tiered dependency structure to minimize installation burden:

- **Core** (`pip install neurosim`): NumPy $\geq$1.24, SciPy $\geq$1.11, Gymnasium $\geq$0.29. All environments and signal models are fully functional with only these dependencies.
- **Data** (`neurosim[data]`): MNE $\geq$1.6, MOABB $\geq$1.0. Required only for loading real EEG datasets.
- **Surrogate** (`neurosim[surrogate]`): PyTorch $\geq$2.0. Required only for the cVAE surrogate model.
- **Training** (`neurosim[train]`): Stable Baselines3 $\geq$2.0, PyTorch $\geq$2.0. Required only for the PPO training script.

This design ensures that importing `neurosim.envs` never triggers PyTorch or MOABB imports, keeping the core package lightweight. PyTorch is imported lazily inside the `NeuralSurrogate` class only when `train()` or `load()` is called.

## 4 Gymnasium Environments

NeuroSim provides three environments that model distinct BCI paradigms. Each environment defines observation spaces, action spaces, reward functions, and episode structures that capture the essential challenges of its paradigm.

### 4.1 DecoderAdapt-v0

The `DecoderAdapt-v0` environment models the motor imagery BCI decoder adaptation problem. The agent receives neural features and must simultaneously classify the imagined movement, decide whether to trigger a costly recalibration, and set the online adaptation rate. This captures the real-world tradeoff where decoder drift degrades accuracy over time, but recalibration interrupts the user's workflow.

**Observation space.** The observation is a dictionary with six components:

- `neural_features` $(24,)$: Band-power and CSP-derived features from EEG, reflecting the current class-conditioned neural activity plus drift and noise.
- `decoder_confidence` $(4,)$: Softmax output of the simulated decoder, providing a probability distribution over the four motor imagery classes.
- `decoder_age` $(1,)$: Normalized count of steps since the last recalibration, indicating how stale the current decoder parameters are.
- `session_progress` $(1,)$: Fraction of the episode elapsed, enabling time-aware policies.
- `recent_accuracy` $(1,)$: Rolling classification accuracy over the last 20 trials.
- `drift_indicator` $(24,)$: Absolute difference between the current feature distribution mean and the calibration distribution mean, serving as an explicit signal of distribution shift.

The total observation dimensionality is 55 when flattened.

**Action space.** The action is a dictionary with three components:

- `classification`: Discrete(4) — the predicted motor imagery class.
- `recalibrate`: Discrete(2) — 0 to keep the current decoder, 1 to trigger recalibration.
- `adaptation_rate`: Box(1) $\in [0, 1]$ — continuous online learning rate controlling how aggressively the decoder adapts to recent data.

For compatibility with Stable Baselines3, the `FlattenDictActionWrapper` converts this into a Box(6) space where discrete actions are encoded as continuous values and rounded back to integers during execution.

**Reward function.**

$$r_t = \underbrace{+1.0}_{\text{correct}} + \underbrace{+0.2 \cdot \mathbb{1}[\text{streak} \geq 5]}_{\text{streak bonus}} - \underbrace{0.3 \cdot \mathbb{1}[\text{recalibrate}]}_{\text{recalibration cost}} \tag{1}$$

The agent receives +1.0 for each correct classification, a +0.2 streak bonus when five or more consecutive classifications are correct, and incurs a −0.3 penalty each time it triggers recalibration.

**Episode structure.** Each episode consists of 300 steps (trials). At each step, a motor imagery class is sampled uniformly from the four classes. The environment simulates feature distribution drift at a configurable rate, shifting the calibration mean away from the current distribution. The episode terminates by truncation after 300 steps.

## 4.2 CursorControl-v0

The `CursorControl-v0` environment models intracortical BCI cursor control, inspired by BrainGate-style systems where a population of motor cortex neurons encodes intended movement direction and speed. The agent decodes neural population activity into 2D velocity commands to acquire screen targets.

**Observation space.** The observation is a dictionary with six components:

- `neural_state` $(96, 5)$: Simulated spike counts from 96 neural units across 5 time bins. Neural activity follows a cosine tuning model where each unit's firing rate depends on the cosine similarity between the intended movement direction and the unit's preferred direction.
- `cursor_position` $(2, )$: Current cursor $[x, y]$ in a normalized $[0, 1] \times [0, 1]$ workspace.
- `target_position` $(2, )$: Target $[x, y]$ coordinates.
- `cursor_velocity` $(2, )$: Current velocity vector.
- `distance_to_target` $(1, )$: Euclidean distance to the target.
- `time_in_trial` $(1, )$: Fraction of maximum trial steps elapsed.

**Action space.** Box(2) $\in [-1, 1]$: a two-dimensional decoded velocity command $[v_x, v_y]$ that displaces the cursor by $\mathbf{v} \cdot 0.05$ per step, clamped to the workspace boundaries.

**Reward function.**

$$r_t = \underbrace{-0.01 \cdot d_t}_{\text{distance}} + \underbrace{1.0 \cdot \Delta d_t}_{\text{progress}} + \underbrace{5.0 \cdot \mathbb{1}[\text{acquire}]}_{\text{acquisition}} - \underbrace{0.1 \cdot \|\mathbf{v}_t - \mathbf{v}_{t-1}\|}_{\text{jitter}} \tag{2}$$

where $d_t$ is the Euclidean distance to the target, $\Delta d_t = d_{t-1} - d_t$ is the progress toward the target, and the jitter penalty discourages erratic velocity changes. The acquisition bonus is scaled by time efficiency: $5.0 \cdot (1 - 0.5 \cdot t_{\text{trial}}/t_{\max})$, rewarding faster reaches.

**Episode structure.** Each episode consists of 50 reach trials. Within each trial, the agent has up to 30 steps to move the cursor within a radius of 0.05 of the target. A trial ends upon target acquisition or timeout, after which a new target is sampled. The cursor resets to the workspace center $(0.5, 0.5)$ at the start of each trial, and targets are sampled uniformly from $[0.1, 0.9]^2$ with a minimum distance of 0.2 from the cursor.

## 4.3 SpellerNav-v0

The `SpellerNav-v0` environment models an active-query BCI speller where the agent controls which stimulus groups to flash and decides when to commit to a symbol selection. This captures the explore–exploit tradeoff in ERP-based BCIs: more flashes improve classification accuracy but reduce the information transfer rate (ITR).

**Observation space.** The observation is a dictionary with five components:

- `posterior` $(36,)$: Bayesian posterior probability distribution over the 36-symbol alphabet (A–Z, 0–9).
- `response_history` $(10, 8)$: The last 10 ERP responses across 8 EEG channels, providing temporal context for evidence accumulation.
- `n_flashes` $(1,)$: Number of flashes used for the current symbol, normalized by the maximum (30).
- `target_queue` $(5,)$: Indices of the upcoming target symbols in the word to be spelled.
- `snr_estimate` $(1,)$: Estimated signal-to-noise ratio computed from the response buffer.

**Action space.** The action is a dictionary with two components:

- `stimulus_group`: Discrete(6) — selects which of the 6 groups (each containing $36/6 = 6$ symbols) to flash.
- `commit`: Discrete(2) — 0 to continue flashing, 1 to commit to the maximum a posteriori (MAP) symbol.

For SB3 compatibility, the `FlattenDictActionWrapper` converts this to MultiDiscrete([6, 2]).

**Reward function.**

$$r_t = \underbrace{-0.02}_{\text{per flash}} + \underbrace{+3.0 \cdot \mathbb{1}[\text{correct commit}]}_{\text{correct}} + \underbrace{(-2.0) \cdot \mathbb{1}[\text{wrong commit}]}_{\text{wrong}} \tag{3}$$

Each flash incurs a small cost of $-0.02$, incentivizing efficient evidence accumulation. A correct commit earns $+3.0$ and an incorrect commit incurs $-2.0$. An additional ITR bonus proportional to $(1 - n_{\text{flashes}}/30)$ rewards faster decisions when correct. If the agent exhausts the maximum of 30 flashes without committing, a forced commit occurs with a reduced correct bonus of $+1.5$.

**Episode structure.** Each episode requires spelling 5 symbols. For each symbol, the agent may flash up to 30 times before a forced commit. The simulated ERP response uses a spatial P300 pattern: when the target symbol is in the flashed group, a positive deflection with spatial gradient $[0.3, 0.5, 0.8, 1.0, 1.0, 0.8, 0.5, 0.3]$ across channels is added, scaled by a configurable amplitude (default 1.5). The posterior is updated via Bayesian inference after each flash.

Table 1: Summary of NeuroSim environment specifications.

| **Property** | **DecoderAdapt-v0** | **CursorControl-v0** | **SpellerNav-v0** |
|---|---|---|---|
| BCI Paradigm | Motor imagery | Intracortical | P300 speller |
| Obs. Dim. (flat) | 55 | 488 | 128 |
| Action Type | Dict (mixed) | Box(2) | Dict (discrete) |
| Episode Length | 300 steps | 50 trials × 30 | 5 symbols × 30 |
| Signal Model | Feature drift | Cosine tuning | P300 + Bayesian |

### 4.4 Environment Summary

Table 1 summarizes the key specifications of all three environments.

## 5 Signal Models

Real BCI signals are affected by multiple sources of non-stationarity that degrade decoder performance over time. NeuroSim models four such sources, each implemented as an independent module that can be composed through the `SignalPipeline` class. The pipeline applies models in a physiologically motivated order: drift models first (electrode and neural degradation), then noise injection (environmental and physiological artifacts), and finally co-adaptation (user learning).

### 5.1 Electrode Impedance Drift

Electrode impedance increases over time due to gel drying, electrode displacement, and skin conductance changes. Higher impedance attenuates the recorded signal amplitude. The `ElectrodeDriftModel` simulates this process:

$$Z(t) = Z_0 \cdot (1 + \alpha_z \cdot t + \epsilon_z(t)) \tag{4}$$

$$\text{signal}_{\text{out}} = \text{signal}_{\text{in}} \cdot \frac{Z_0}{Z(t)} \tag{5}$$

where $\alpha_z$ is the linear drift rate per timestep (default 0.001), $\epsilon_z(t) \sim \mathcal{N}(0, \sigma_z^2)$ is per-channel impedance noise with $\sigma_z = 0.01$, and $Z_0$ is the initial impedance. The impedance noise is sampled independently for each channel, reflecting the physical reality that different electrodes drift at different rates. A floor of $Z(t)/Z_0 \geq 0.01$ prevents numerical instability.

### 5.2 Neural Fatigue

Extended BCI use causes neural fatigue, reducing the strength and discriminability of neural signals. The `FatigueDriftModel` follows an exponential saturation curve:

$$f(t) = 1 - \beta_f \cdot \left(1 - e^{-t/\tau_f}\right) \tag{6}$$

$$\text{signal}_{\text{out}} = \text{signal}_{\text{in}} \cdot f(t) \tag{7}$$

where $\beta_f \in [0, 1]$ controls the maximum fatigue effect (default 0.3, corresponding to a 30% signal reduction at asymptote), and $\tau_f$ is the time constant controlling fatigue onset speed (default 100 timesteps). As $t \to \infty$, $f(t) \to 1 - \beta_f$. Small Gaussian noise (std = 0.02) is added to the fatigue factor to model trial-to-trial variability, and the factor is clamped to $[0.01, 1.0]$.

## 5.3 Feature Distribution Shift

The statistical properties of neural features drift over time due to changes in mental strategy, arousal, and environmental conditions. The `FeatureShiftModel` combines continuous linear drift with stochastic step shifts:

$$\mu_{\text{shifted}}(t) = \mu_0 + r_{\text{drift}} \cdot t + \sum_k s_k \tag{8}$$

where $r_{\text{drift}}$ is the continuous drift rate per timestep (default 0.0001), and $s_k$ are discrete step shifts that occur independently at each timestep with probability $p_{\text{step}}$ (default 0.01). Each step shift is drawn from $\mathcal{N}(0, \sigma^2_{\text{step}})$ with $\sigma_{\text{step}} = 0.5$. The step shifts model sudden distribution changes such as a subject adjusting their seating position or shifting mental strategy. Step shifts accumulate additively over the course of an episode.

## 5.4 User–Decoder Co-Adaptation

In closed-loop BCI use, users gradually adapt their neural strategies to produce signals that the decoder can classify more easily. This co-adaptation is a critical real-world phenomenon that both improves and constrains long-term BCI performance. The `CoAdaptationModel` simulates the user's side of this interaction:

$$\alpha(t) = \alpha_{\text{base}} \cdot \left(1 - e^{-t/\tau_{\text{adapt}}}\right) \tag{9}$$

$$\text{adapted}(t) = (1 - \alpha(t)) \cdot \text{signal} + \alpha(t) \cdot \text{decoder\_expectation} \tag{10}$$

where $\alpha(t)$ is the adaptation strength that increases over time and saturates at $\alpha_{\text{base}}$. The base adaptation rate is sampled uniformly from a configurable range (default $[0.1, 0.5]$) to model inter-subject variability, and $\tau_{\text{adapt}}$ (default 50) controls the speed of adaptation onset. Small Gaussian noise (std = 0.02) is added to $\alpha(t)$ to model imperfect human learning. The model interpolates between the user's natural signal and the decoder's expected signal pattern, effectively simulating how users learn to produce more "decoder-friendly" neural patterns over time.

## 5.5 Noise Injection

In addition to the drift and co-adaptation models, NeuroSim provides a `NoiseInjector` module that adds four types of physiological and environmental noise observed in real EEG recordings:

- **Pink (1/f) noise**: Generated by applying $1/\sqrt{f}$ scaling to white noise in the frequency domain. Pink noise matches the spectral profile of background EEG activity.
- **EMG artifacts**: High-frequency broadband bursts modeling muscle tension contamination, occurring at each timepoint with a configurable probability.
- **Eye blink artifacts**: Large-amplitude slow deflections modeled as Gaussian-enveloped waves (150–400 ms duration, 3–8$\times$ signal RMS amplitude) with a frontal-to-posterior attenuation gradient.
- **Line noise**: 50/60 Hz sinusoidal interference with per-channel random phase.

Noise severity is controlled through three presets (`low`, `medium`, `high`) that scale the amplitude and probability parameters of each noise type.

### 5.6 Signal Pipeline Composition

The `SignalPipeline` class chains all signal models into a single callable unit. Models are applied in order: drift models first, then noise, then co-adaptation. This ordering matches the real signal corruption chain where neural signals are first degraded by physical electrode changes, then corrupted by environmental noise, and finally modified by user learning.

The pipeline supports four difficulty presets:

- **none**: No signal corruption.
- **mild**: Electrode drift only ($\alpha_z = 0.0005$, $\sigma_z = 0.005$).
- **moderate**: Electrode drift + fatigue + medium noise.
- **full**: All drift models + noise + co-adaptation.

Environments accept a `signal_pipeline` parameter, allowing researchers to study agent robustness under controlled non-stationarity.

## 6 Neural Surrogate

Training RL agents requires large volumes of neural data that may not be available from real recordings. NeuroSim includes a conditional Variational Autoencoder (cVAE) that learns the distribution of real EEG epochs and generates unlimited synthetic training data conditioned on class labels.

### 6.1 Architecture

The `NeuralSurrogate` implements a cVAE with the following architecture:

**Encoder:**

$$\mathbf{h} = \text{ReLU}\left(\text{BN}\left(\text{ReLU}\left(\text{BN}\left(\text{Linear}_{C\cdot T+K\to 512}([\mathbf{x};\mathbf{c}])\right)\right)\to 256\right)\right)\to 128 \tag{11}$$

$$\boldsymbol{\mu} = \text{Linear}_{128\to d}(\mathbf{h}), \quad \log \boldsymbol{\sigma}^2 = \text{Linear}_{128\to d}(\mathbf{h}) \tag{12}$$

where $\mathbf{x} \in \mathbb{R}^{C\cdot T}$ is the flattened EEG signal, $\mathbf{c} \in \mathbb{R}^K$ is the one-hot class label, $d = 32$ is the latent dimension, and BN denotes batch normalization.

**Reparameterization:**

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{13}$$

**Decoder:**

$$\hat{\mathbf{x}} = \text{Linear}_{512\to C\cdot T}\left(\text{ReLU}\left(\text{BN}\left(\text{ReLU}\left(\text{BN}\left(\text{Linear}_{d+K\to 128}([\mathbf{z};\mathbf{c}])\right)\right)\to 256\right)\right)\to 512\right) \tag{14}$$

The decoder mirrors the encoder structure: $[d + K] \to 128 \to 256 \to 512 \to [C \cdot T]$, with ReLU activations and batch normalization at each intermediate layer.

### 6.2 Training

The cVAE is trained to minimize the standard VAE objective:

$$\mathcal{L} = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|^2}_{\text{MSE reconstruction}} + \underbrace{\beta \cdot \text{KL}\left(q(\mathbf{z}|\mathbf{x}, \mathbf{c}) \| p(\mathbf{z})\right)}_{\text{KL divergence}} \tag{15}$$

where the KL divergence term has the closed-form expression:

$$\mathrm{KL} = -\frac{1}{2}\sum_{j=1}^{d}\left(1 + \log\sigma_j^2 - \mu_j^2 - \sigma_j^2\right) \tag{16}$$

The model is optimized with Adam (learning rate $10^{-3}$) and supports $\beta$-VAE weighting through a configurable `kl_weight` parameter. Training accepts a list of `NeuralEpoch` objects and automatically infers the number of channels, timepoints, and classes from the data.

### 6.3 Generation

After training, the surrogate generates synthetic epochs by sampling from the prior $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, concatenating a one-hot class condition, and decoding. The generated signals are reshaped to $(C \times T)$ and wrapped in `NeuralEpoch` objects with appropriate metadata marking them as synthetic. Each generated epoch inherits the channel names and sampling frequency from the training data.

### 6.4 Persistence

The model supports save/load operations via PyTorch's serialization. The saved state includes all network weights, architectural configuration (channels, timepoints, latent dimension, classes), channel names, sampling frequency, and the full training loss history. An untrained model raises a `RuntimeError` when `generate()` or `save()` is called, preventing accidental use of uninitialized parameters.

## 7 Experiments

We evaluate PPO [Schulman et al., 2017] against a random baseline across all three NeuroSim environments to establish initial performance benchmarks.

### 7.1 Experimental Setup

All experiments use Stable Baselines3's PPO implementation with default hyperparameters. Each agent is trained for 20,000 timesteps, followed by evaluation over 20 episodes. A random agent that samples uniformly from each environment's action space serves as the baseline. All experiments use a fixed random seed of 42 for reproducibility. Training was conducted on a consumer CPU; wall-clock times ranged from 6.57 to 8.37 seconds per environment.

For environments with Dict action spaces, the `FlattenDictActionWrapper` is applied before training. The `ObservationFlattener` is applied by SB3's internal observation handling.

### 7.2 Results

Table 2 presents the evaluation results.

Table 2: PPO vs. random agent evaluation results (20,000 training timesteps, 20 evaluation episodes). Mean ± standard deviation of episode returns.

| **Environment** | **PPO** | **Random** | **Improvement** |
|---|---|---|---|
| DecoderAdapt-v0 | $121.11 \pm 20.66$ | $23.70 \pm 7.60$ | $+97.41$ |
| CursorControl-v0 | $-89.99 \pm 27.71$ | $-209.21 \pm 8.53$ | $+119.22$ |
| SpellerNav-v0 | $-11.60 \pm 2.04$ | $-5.55 \pm 2.79$ | $-6.05$ |

Figure 2 shows the episode reward progression during PPO training across all three environments. The smoothed curves illustrate learning dynamics: DecoderAdapt-v0 shows steady improvement above the random baseline, CursorControl-v0 shows gradual improvement from a low starting point, and SpellerNav-v0 shows the policy struggling to exceed random performance within 20,000 timesteps.

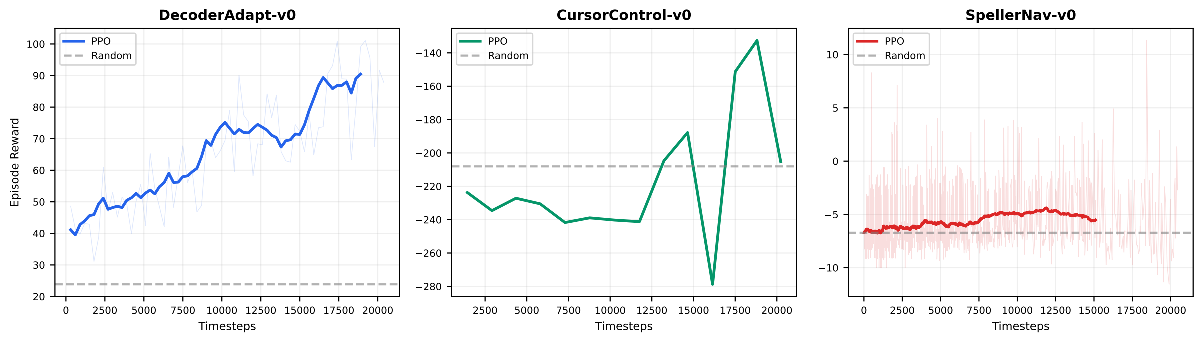

Figure 2: PPO training reward progression over 20,000 timesteps for each NeuroSim environment. Light traces show individual episode rewards; bold curves are smoothed moving averages. Dashed gray lines indicate the random baseline mean. DecoderAdapt-v0 and CursorControl-v0 show clear learning; SpellerNav-v0 requires more training timesteps.

## 7.3 Analysis

**DecoderAdapt-v0.** PPO achieves a mean reward of 121.11, a 5.1× improvement over the random baseline (23.70). This indicates that the agent has learned to classify motor imagery classes with reasonable accuracy and has developed a strategy for managing the recalibration–accuracy tradeoff. The moderate standard deviation (20.66) reflects variability across episodes due to stochastic drift and class sampling.

**CursorControl-v0.** PPO achieves −89.99 compared to −209.21 for random, an improvement of 119.22 reward points. The negative absolute reward reflects the continuous distance penalty: even an optimal agent accumulates negative reward from the $-0.01 \cdot d_t$ term during each reach. The substantial improvement indicates that PPO has learned to move the cursor toward targets rather than randomly, though the moderate budget of 20,000 timesteps limits the quality of the learned velocity control policy.

**SpellerNav-v0.** PPO underperforms the random baseline (−11.60 vs. −5.55, a deterioration of 6.05 points). This result, while initially surprising, has a clear explanation. The SpellerNav environment requires the agent to learn two distinct skills: (1) selecting informative stimulus groups that maximally reduce posterior entropy, and (2) timing the commit decision to balance accuracy against flash cost. At 20,000 timesteps, PPO has insufficient experience to discover the relationship between stimulus selection, posterior evolution, and commit timing. The random agent benefits from uniform exploration across all stimulus groups, which provides adequate evidence accumulation through sheer coverage, while PPO's early policy may fixate on suboptimal stimulus groups before gathering enough reward signal to correct course.

This result highlights an important property of the SpellerNav environment: it demands more exploration than the other two environments because the reward signal for stimulus selection is indirect (mediated through the posterior update and eventual commit outcome). We expect PPO to surpass the random baseline with significantly more training timesteps (e.g., 200K+) or with exploration-encouraging techniques such as entropy bonus tuning or curiosity-driven exploration.

### 7.4 Classical Baseline: CSP+LDA

To contextualize the RL results, NeuroSim includes a classical BCI decoding pipeline: Common Spatial Patterns (CSP) for spatial filtering combined with Fisher's Linear Discriminant Analysis (LDA) for classification. The implementation uses only NumPy and SciPy, avoiding an sklearn dependency.

The CSP+LDA baseline operates on `NeuralEpoch` objects, fitting spatial filters from training epochs and extracting log-variance features for LDA classification. It supports binary classification (two motor imagery classes) and provides a `fit`/`predict`/`evaluate` interface consistent with the RL agent API. This baseline represents the traditional BCI decoding approach that RL agents should ultimately outperform, particularly under non-stationary conditions where static classifiers cannot adapt.

Table 3 reports within-subject classification accuracy on synthetic 22-channel motor imagery epochs with lateralized event-related desynchronization (ERD) at $\mathrm{SNR} = 0.15$, volume conduction via a random mixing matrix, and 50 Hz line noise. Ten random seeds control epoch generation; 200 epochs per seed are split 50/50 for training and evaluation.

Table 3: CSP+LDA within-subject classification accuracy on synthetic 22-channel motor imagery epochs ($\mathrm{SNR} = 0.15$, 10 seeds, 200 epochs each).

| Metric | Value |
|---|---|
| Mean accuracy | 86.2% |
| Std. | ± 5.0% |
| Min (seed) | 75.0% |
| Max (seed) | 92.5% |
| Channels | 22 |
| CSP components | 6 |

### 7.5 Cross-Subject Transfer Evaluation

Generalization across subjects is a critical challenge in BCI research, as inter-subject variability in neural signals often necessitates per-subject calibration. NeuroSim provides a leave-one-subject-out (LOSO) cross-validation protocol for evaluating cross-subject transfer.

The protocol generates a synthetic cohort of subjects, each with subject-specific parameters (signal-to-noise ratio, spatial pattern strength, and channel count variation) drawn from controlled distributions. For each fold, a CSP+LDA classifier is trained on all subjects except one and evaluated on the held-out subject. The `TransferResult` dataclass records per-subject accuracy, enabling analysis of which subject characteristics predict transferability.

This protocol serves as a foundation for evaluating RL agent generalization: agents trained with the cVAE surrogate on multi-subject data can be assessed against the CSP+LDA transfer baseline to quantify the benefit of adaptive decoding strategies across subject boundaries.

Table 4 reports leave-one-subject-out accuracy for a nine-subject synthetic cohort (22 channels, 100 epochs per subject). Per-subject SNR is drawn from the range $[0.08, 0.22]$ to simulate realistic inter-subject variability; subject 3 has the lowest SNR and correspondingly lowest accuracy.

## 8 Discussion

### 8.1 Design Tradeoffs

NeuroSim navigates several fundamental design tensions:

Table 4: Leave-one-subject-out (LOSO) cross-validation accuracy using CSP+LDA on a 9-subject synthetic cohort (22 channels, 100 epochs/subject).

| Subject | Accuracy |
|---|---|
| S1 | 93% |
| S2 | 95% |
| S3 | 83% |
| S4 | 92% |
| S5 | 97% |
| S6 | 94% |
| S7 | 97% |
| S8 | 95% |
| S9 | 93% |
| Mean ± Std | 93.2% ± 4.0% |

**Fidelity vs. speed.** High-fidelity neural simulation requires detailed biophysical models (e.g., Hodgkin-Huxley neurons, volume conduction), but these are computationally expensive and unnecessary for RL training at scale. NeuroSim uses phenomenological models (impedance drift, fatigue curves, cosine tuning) that capture the statistical properties relevant to decoder performance without modeling the underlying biophysics. This tradeoff enables fast simulation—training 20,000 timesteps takes under 9 seconds on a consumer CPU—while preserving the signal characteristics that challenge real BCI decoders.

**Generality vs. specificity.** Each BCI paradigm has unique signal characteristics and control requirements. Rather than building a single monolithic environment, NeuroSim provides three specialized environments that each capture the essential challenges of their paradigm. The shared signal pipeline and data infrastructure provide cross-paradigm consistency while allowing paradigm-specific observation and action spaces.

**Dict vs. flat spaces.** Gymnasium's Dict spaces enable semantically rich observations (e.g., separate fields for neural features, confidence, and drift indicators) and structured actions (e.g., simultaneous classification and recalibration decisions). However, many RL algorithms require flat vector spaces. NeuroSim supports both through wrapper classes, prioritizing semantic clarity in the native environment interface while providing compatibility layers for practical training.

## 8.2 Extensibility

NeuroSim is designed for extension along several axes:

**New environments.** The signal pipeline and data infrastructure are environment-agnostic. Adding a new BCI paradigm requires only implementing a new Gymnasium environment class and registering it under the `neurosim/` namespace.

**New signal models.** Drift models inherit from `BaseDriftModel` and implement a single `apply(signal, timestep)` method. New models (e.g., electrode delamination, circadian rhythm effects) can be added and composed through the `SignalPipeline` without modifying existing code.

**Real data integration.** The `MOABBLoader` supports 11+ datasets and can be extended to additional MOABB datasets or custom data sources by implementing the `NeuralEpoch` interface. The cVAE surrogate can be trained on any dataset to generate paradigm-specific synthetic data.

## 8.3 Sim-to-Real Gap

The primary limitation of any simulation-based approach is the gap between simulated and real neural signals. NeuroSim's signal models are grounded in established BCI literature, but several aspects of real neural variability are not yet captured:

- **Inter-subject variability**: While the co-adaptation model samples subject-specific adaptation rates, the underlying neural activity generation uses simple parametric models rather than subject-specific distributions.
- **Closed-loop effects**: Real BCI users receive visual feedback that influences their subsequent neural activity. NeuroSim's co-adaptation model approximates this feedback loop but does not fully capture the dynamics of real-time visual-motor interaction.
- **Non-stationary noise**: Real artifact patterns (eye blinks, muscle tension) are correlated with task demands and user state. NeuroSim's noise models use stationary stochastic processes.
- **Neural population dynamics**: The CursorControl environment's cosine tuning model captures first-order directional tuning but does not model higher-order neural dynamics such as preparatory activity, neural adaptation, or population-level correlations.

The cVAE surrogate partially addresses the sim-to-real gap by learning signal distributions from real recordings, but its effectiveness depends on the quantity and diversity of the training data.

## 8.4 Limitations

Beyond the sim-to-real gap, NeuroSim has several current limitations:

1. The cVAE surrogate uses a fully connected architecture that does not exploit the temporal or spatial structure of EEG signals. Convolutional or recurrent architectures would likely improve generation quality.
2. The SpellerNav environment's Bayesian posterior update uses a simplified likelihood model. A more principled approach would use discriminative models trained on real ERP data.
3. The benchmark results use only PPO with default hyperparameters. Comprehensive benchmarking across multiple RL algorithms and hyperparameter configurations remains future work.
4. The environments do not currently support multi-agent scenarios where multiple decoders or users interact simultaneously.
5. All experiments use a single random seed (42) for reproducibility. While per-seed variation is reported for CSP+LDA (10 seeds) and per-subject variation for LOSO (9 subjects), the overall experimental configuration has not been validated across multiple independent seeds. Multi-seed replication across the full benchmark suite is future work.

## 8.5 Software Quality

NeuroSim includes a comprehensive test suite of 158 tests covering all modules. The tests execute in under 12 seconds without requiring network access, GPU hardware, or MOABB downloads. Tests validate signal model correctness (drift attenuation, fatigue curves, step shift accumulation), environment compliance (observation and action space shapes, reward computation, episode truncation, render output, seed determinism), data pipeline integrity (feature extraction shapes and value ranges, preprocessing ordering), surrogate behavior (training loss convergence, class-conditioned generation, save/load roundtrips), wrapper correctness (Dict-to-Box and Dict-to-MultiDiscrete conversion, passthrough for compatible spaces), agent baselines (random agent evaluation, CSP+LDA fit, predict, and evaluate with synthetic epochs), benchmark metrics (classification accuracy, information transfer rate, reach time, path efficiency, adaptation efficiency), and cross-subject transfer (cohort generation, parameter ranges, LOSO protocol execution). All tests mock MOABB access to ensure offline execution.

The codebase uses Python 3.10+ type annotations throughout, enforces style consistency via Ruff, and supports type checking with mypy.

## 9 Conclusion

We have presented NeuroSim, a Gymnasium-compatible reinforcement learning environment suite for brain-computer interfaces. The platform provides three environments spanning the primary BCI paradigms—motor imagery decoder adaptation, continuous cursor control, and ERP-based speller navigation—each incorporating physiologically grounded signal models for non-stationarity simulation. A conditional VAE surrogate enables unlimited synthetic data generation from real or synthetic training epochs.

Baseline PPO experiments demonstrate that standard RL algorithms can learn meaningful BCI control policies in two of the three environments within a modest training budget of 20,000 timesteps, while the more challenging SpellerNav environment requires additional exploration to overcome the indirect reward structure. A classical CSP+LDA baseline and a leave-one-subject-out cross-subject transfer protocol provide additional evaluation axes for comparing adaptive RL agents against traditional decoding pipelines.

NeuroSim is available as an installable Python package (`pip install neurosim`) under the MIT license, with source code, trained models, and benchmark infrastructure at `https://github.com/HassDhia/neurosim`. The package requires only NumPy, SciPy, and Gymnasium as core dependencies, with optional extras for real dataset loading (MOABB, MNE), surrogate training (PyTorch), and RL agent training (Stable Baselines3).

Future work includes expanding the environment suite to SSVEP and hybrid BCI paradigms, implementing convolutional and recurrent cVAE architectures for higher-fidelity surrogate generation, extending the cross-subject transfer protocol to evaluate sim-to-real policy transfer with real EEG recordings, and conducting comprehensive multi-algorithm comparisons across extended training budgets. We anticipate that NeuroSim will serve as a foundation for reproducible RL research in brain-computer interfaces, lowering the barrier to entry for researchers who lack access to expensive neural recording equipment.

## Acknowledgments

The author thanks the MOABB development team for maintaining the open BCI dataset ecosystem, the Gymnasium and Stable Baselines3 communities for the RL infrastructure that NeuroSim builds upon, and the broader open-source BCI research community for the datasets and standards that make this work possible.

## References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint*, 2016.

Vinay Jayaram and Alexandre Barachant. MOABB: trustworthy algorithm benchmarking for BCIs. *Journal of Neural Engineering*, 15(6):066011, 2018. doi: 10.1088/1741-2552/aadea0.

Jonathan C. Kao, Sergey D. Stavisky, David Sussillo, Paul Nuyujukian, and Krishna V. Shenoy. AI copilots for brain–computer interfaces. *Nature Machine Intelligence*, 7(1):57–69, 2025. doi: 10.1038/s42256-024-00966-x.

Xuan Liang and Jonathan C. Kao. A brain–computer interface simulator for reinforcement learning agent training. *bioRxiv*, 2024. doi: 10.1101/2024.05.13.593955. Preprint, not peer-reviewed.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dorber. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Justin C. Sanchez, Babak Mahmoudi, Jack DiGiovanna, and Jose C. Principe. Exploiting co-adaptation for the design of symbiotic neuroprosthetic assistants. *Neural Networks*, 22(3): 305–315, 2009. doi: 10.1016/j.neunet.2009.03.015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint*, 2017.

Yueran Zhang, Xuan Liang, and Jonathan C. Kao. CopAC: Co-adaptive policy actor-critic for brain–computer interfaces. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 58932–58948. PMLR, 2024.

## A Installation and Quick Start

NeuroSim requires Python 3.10 or later. Installation from PyPI:

```
pip install neurosim              # core only
pip install neurosim[all]         # all optional dependencies
```

Minimal usage example:

```
import gymnasium as gym

env = gym.make("neurosim/DecoderAdapt-v0")
obs, info = env.reset(seed=42)

for step in range(300):
    action = env.action_space.sample()
    obs, reward, terminated, truncated, info = env.step(action)
    if terminated or truncated:
        break

env.close()
```

Training with PPO via the command-line interface:

```
neurosim-train                    # trains PPO on all environments
neurosim-benchmark                # evaluates agents on benchmark suite
```

## B Signal Model Default Parameters

Table 5 lists the default parameters for each signal model.

Table 5: Default parameters for NeuroSim signal models.

| **Model** | **Parameter** | **Default** | **Description** |
|---|---|---|---|
| ElectrodeDrift | $\alpha_z$ | 0.001 | Linear drift rate |
| | $\sigma_z$ | 0.01 | Impedance noise std |
| FatigueDrift | $\beta_f$ | 0.3 | Maximum fatigue effect |
| | $\tau_f$ | 100.0 | Time constant (steps) |
| | noise_std | 0.02 | Per-trial noise |
| FeatureShift | $r_{\text{drift}}$ | 0.0001 | Continuous drift rate |
| | $p_{\text{step}}$ | 0.01 | Step shift probability |
| | $\sigma_{\text{step}}$ | 0.5 | Step shift magnitude |
| CoAdaptation | $\alpha_{\text{base}}$ | $\mathcal{U}(0.1, 0.5)$ | Base adaptation rate |
| | $\tau_{\text{adapt}}$ | 50.0 | Adaptation time constant |

## C cVAE Architecture Summary

Table 6 summarizes the cVAE network architecture.

Table 6: cVAE network architecture. $C$: channels, $T$: timepoints, $K$: number of classes, $d$: latent dimension (32).

| **Component** | **Layers** | **Output Dim** |
|---|---|---|
| Encoder | Linear($C{\cdot}T + K$, 512) + ReLU + BN | 512 |
| | Linear(512, 256) + ReLU + BN | 256 |
| | Linear(256, 128) + ReLU | 128 |
| Latent $\boldsymbol{\mu}$ | Linear(128, $d$) | 32 |
| Latent $\log \boldsymbol{\sigma}^2$ | Linear(128, $d$) | 32 |
| Decoder | Linear($d + K$, 128) + ReLU + BN | 128 |
| | Linear(128, 256) + ReLU + BN | 256 |
| | Linear(256, 512) + ReLU | 512 |
| | Linear(512, $C{\cdot}T$) | $C{\cdot}T$ |