A Supplementary Materials

A.1 Cell Culture

Neural cells were cultured either from the cortices of E15.5 mouse embryos or differentiated from human induced pluripotent stem cells via a dual SMAD inhibition (DSI) protocol or through a lentivirus based NGN2 direct differentation protocols as previously described [13]. Cells were cultured until plating. For primary mouse neurons this occurred at day-in-vitro (DIV) 0, for DSI cultures this occurred at between DIV 30 - 33 depending culture development, for NGN2 cultures this occured at DIV 3.

A.2 MEA Setup and Plating

MaxOne Multielectrode Arrays (MEA; Maxwell Biosystems, AG, Switzerland) was used and is a high-resolution electrophysiology platform featuring 26,000 platinum electrodes arranged over an 8 mm2. The MaxOne system is based on complementary meta-oxide-semiconductor (CMOS) technology and allows recording from up to 1024 channels. MEAs were coated with either polyethylenimine (PEI) in borate buffer for primary culture cells or Poly-D-Lysine for cells from an iPSC background before being coated with either 10 µg/ml mouse laminin or 10 µg/ml human 521 Laminin (Stemcell Technologies Australia, Melbourne, Australia) respectively to facilitate cell adhesion.Approximately 10^6 cells were plated on MEA after preparation as per [13]. Cells were allowed approximately one hour to adhere to MEA surface before the well was flooded. The day after plating, cell culture media was changed for all culture types to BrainPhys™ Neuronal Medium (Stemcell Technologies Australia, Melbourne, Australia) supplemented with 1% penicillin-streptomycin. Cultures were maintained in a low O2 incubator kept at 5% CO2, 5% O2, 36°C and 80% relative humidity. Every two days, half the media from each well was removed and replaced with free media. Media changes always occurred after all recording sessions.

A.3 DishBrain platform and electrode configuration

The current DishBrain platform is configured as a low-latency, real-time MEA control system with on-line spike detection and recording software. The DishBrain platform provides on-line spike detection and recording configured as a low-latency, real-time MEA control. The DishBrain software runs at 20 kHz and allows recording at an incredibly fine timescale. There is the option of recording spikes in binary files, and regardless of recording, they are counted over a period of 10 milliseconds (200 samples), at which point the game environment is provided with how many spikes are detected in each electrode in each predefined motor region as described below. Based on which motor region the spikes occurred in, they are interpreted as motor activity, moving the 'paddle' up or down in the virtual space. As the ball moves around the play area at a fixed speed and bounces off the edge of the play area and the paddle, the pong game is also updated at every 10ms interval. Once the ball hits the edge of the play area behind the paddle, one rally of pong has come to an end. The game environment will instead determine which type of feedback to apply at the end of the rally: random, silent, or none. Feedback is also provided when the ball contacts the paddle under the standard stimulus condition. A 'stimulation sequencer' module tracks the location of the ball relative to the paddle during each rally and encodes it as stimulation to one of eight stimulation sites. Each time a sample is received from the MEA, the stimulation sequencer is updated 20,000 times a second, and after the previous lot of MEA commands has completed, it constructs a new sequence of MEA commands based on the information it has been configured to transmit based on both place codes and rate codes. The stimulations take the form of a short square bi-phasic pulse that is a positive voltage, then a negative voltage. This pulse sequence is read and applied to the electrode by a Digital to Analog Converter (or DAC) on the MEA. A real-time interactive version of the game visualiseris available at https://spikestream.corticallabs.com/. Alternatively, cells could be recorded at 'rest' in a gameplay environment where activity was recorded to move the paddle but no stimulation was delivered, with corresponding outcomes still recorded. Using this spontaneous activity alone as a baseline, the gameplay characteristics of a culture were determined. Low level code for interacting with Maxwell API was written in C to minimize processing latencies-so packet processing latency was typically <50 μs. High-level code was written in Python, including configuration setups and general instructions for game settings. A 5 ms spike-to-stim latency was achieved, which was sub-

stantially due to MaxOne's inflexible hardware buffering. Figure S1 illustrates a schematic view of Software components and data flow in the DishBrain closed loop system.

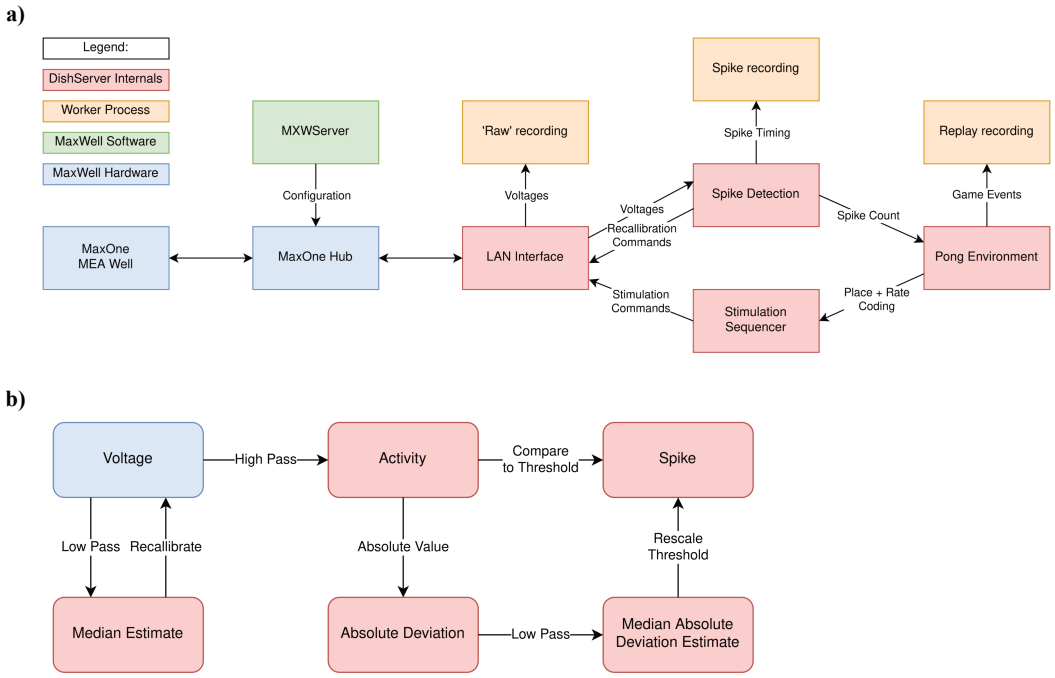

Figure S1: a, b) Schematics of software used for DishBrain. **a)** Software components and data flow in the DishBrain closed loop system. Voltage samples flow from the MEA to the 'Pong' environment, and sensory information flows from the 'Pong' environment back to the MEA, forming a closed loop. The blue rectangles mark proprietary pieces of hardware from MaxWell, including the MEA well which may contain a live culture of neurons. The green MXWServer is a piece of software provided by MaxWell which is used to configure the MEA and Hub, using a private API directly over the network. The red rectangles mark components of the 'DishServer' program, a high-performance program consisting of four components designed to run asynchronously, despite being run on a single CPU thread. The 'LAN Interface' component stores network state, for talking to the Hub, and produces arrays of voltage values for processing. Voltage values are passed to the 'Spike Detection' component, which stores feedback values and spike counts, and passes recalibration commands back to the LAN Interface. When the pong environment is ready to run, it updates the state of the paddle based on the spike counts, updates the state of the ball based on its velocity and collision conditions, and reconfigures the stimulation sequencer based on the relative position of the ball and current state of the game. The stimulation sequencer stores and updates indices and countdowns relating to the stimulations it must produce and converts these into commands each time the corresponding countdown reaches zero, which are finally passed back to the LAN Interface, to send to the MEA system, closing the loop. The procedures associated with each component are run one after the other in a simple loop control flow, but the 'Pong' environment only moves forward every 200th update, short-circuiting otherwise. Additionally, up to three worker processes are launched in parallel, depending on which parts of the system need to be recorded. They receive data from the main thread via shared memory and write it to file, allowing the main thread to continue processing data without having to hand control to the operating system and back again. **b)** Numeric operations in the real-time spike detection component of the DishBrain closed loop system, including multiple IIR filters. Running a virtual environment in a closed loop imposes strict performance requirements, and digital signal processing is the main bottleneck of this system, with close to 42 MB of data to process every second. Simple sequences of IIR digital filters is applied to incoming data, storing multiple arrays of 1024 feedback values in between each sample. First, spikes on the incoming data are detected by applying a high pass filter to determine the deviation of the activity, and comparing that to the MAD, which is itself calculated with a subsequent low pass filter. Then, a low pass filter is applied to the original data to determine whether the MEA hardware needs to be re-calibrated, affecting future samples. This system was able to keep up with the incoming data on a single thread of an Intel Core i7-8809G. Figures adapted from [13].

A.4 Deep Reinforcement Learning Algorithms

Deep Q Network (DQN): The utilized DQN algorithm begins by extracting spatiotemporal features from inputs, such as the movement of the ball in game of 'Pong'. Multiple fully connected layers are used to process the final feature map, which implicitly encodes the effects of actions. As opposed to traditional controllers that use fixed preprocessing steps, this method can adapt processing of the state based on changes in the learning signal.

Algorithm 1 Deep Q Network (DQN) with Experience Replay

Require:
$\mathcal{D}$: Replay memory with capacity N (Default: 10000)
θ: Initial network parameters
$\tilde{\theta}$: Copy of θ
N_b: Training batch size (Default: 5)
$\tilde{N}$: Target network update frequency (Default: 10)
x_t: Input matrix at time t
S: Number of seeds (Default: 40)
e_{max}: Maximum number of episodes (Default: 70)
for seed $\in \{1, \cdots, S\}$ **do**
  for episode $e \in \{1, \cdots, e_{max}\}$ **do**
    Set state $s_1 \leftarrow x_1$ and preprocess $\phi_1 = \phi(s_1)$
    $t = 1$
    while ϕ_t is non-terminal **do**
      With probability ϵ select a random action a_t
      otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
      Execute action a_t and observe reward r_t and input x_{t+1}
      Set new state s_{t+1} and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
      Sample random minibatch of N_b transitions$(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
      Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
      Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
      Replace target parameters $\tilde{\theta} \leftarrow \theta$ every $\tilde{N}$ steps
      $t = t + 1$
    end while
  end for
end for

Advantage Actor-Critic (A2C): In an A2C model, the total reward itself could be represented as a *value* of the state plus the advantage of the action. The *value* of each policy is learned while following it. The policy gradient can be calculated by knowing the *value* for any state. The policy network is then updated such that the probability of actions with a higher advantage values is increased. Here, the policy network (which returns a probability distribution of actions) is called the *actor*, as it tells the agents what to do. *Critic* is another network which enables the evaluation of the actions to decide whether they were good or not. In this case, policy and value are implemented as separate heads of the network, which transform the output from the common body into either probability distributions or single numbers representing the state's value. Thus, low-level features can be shared between the two networks.

Proximal Policy Optimization (PPO): PPO models are a family of policy gradient methods for reinforcement learning. The PPO method uses a slightly different training procedure: An extended set of samples is taken from the environment, and then the advantage is estimated for the whole set or sequence of samples before several epochs of training are performed To estimate policy gradients, instead of using the gradient of action probabilities, the PPO method uses a different objective: the ratio between the new and the old policy scaled by the advantages.

Algorithm 2 Advantage Actor-Critic (A2C)

Require:
θ_v: Initial parameter vector for the value net (critic)
θ_π: Initial parameter vector for the policy net (actor)
N: Number of consecutive steps to play current policy in the environment (Default: 5)
x_t: Input matrix at time t
S: Number of seeds (Default: 40)
e_{max}: Maximum number of episodes (Default: 70)
for seed $\in \{1, \cdots, S\}$ **do**
  $t = 1$
  $e = 1$
  repeat
    $\partial\theta_\pi \leftarrow 0$ and $\partial\theta_v \leftarrow 0$
    $t_{start} = t$
    Set state $s_t \leftarrow x_t$ and preprocess $\phi_t = \phi(s_t)$
    repeat
      Select a_t according to $\pi(a_t|\phi_t;\theta)$
      Execute action a_t and observe reward r_t and input x_{t+1}
      Set new state s_{t+1} and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      $t \leftarrow t + 1$
    until ϕ_t is terminal **or** $t - t_{start} = N$
    $R = \begin{cases} 0 & \text{for terminal } \phi_t \\ V(\phi_t;\theta_v) & \text{for non-terminal } \phi_t \end{cases}$
    for $i \in \{t-1, \cdots, t_{start}\}$ **do**
      $R \leftarrow r_i + \gamma R$
      Accumulate the policy gradients: $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi(a_i|\phi_i;\theta)\big(R - V(\phi_i,\theta_v)\big)$
      Accumulate the value gradients: $\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial\big(R - V(\phi_i,\theta_v)\big)^2}{\partial\theta_v}$
    end for
    Update θ_π and θ_v using $\partial\theta_\pi$ and $\partial\theta_v$, respectively.
    if ϕ_t is terminal **then**
      $e \leftarrow e + 1$
    end if
  until $e > e_{max}$
end for

Algorithm 3 Proximal Policy Optimization (PPO)

Require:
θ_0: Initial policy parameter vector
ϵ: Clipping threshold (Default: 0.2)
N: Number of consecutive steps to play current policy in the environment (Default: 5)
x_t: Input matrix at time t
S: Number of seeds (Default: 40)
e_{max}: Maximum number of episodes (Default: 70)
for seed $\in \{1, \cdots, S\}$ **do**
  $t = 1$
  $e = 1$
  repeat
    $t_{start} = t$
    Set state $s_t \leftarrow x_t$ and preprocess $\phi_t = \phi(s_t)$
    repeat
      Select a_t according to $\pi(a_t|\phi_t;\theta)$
      Execute action a_t and observe reward r_t and input x_{t+1}
      Set new state s_{t+1} and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      $t \leftarrow t + 1$
    until ϕ_t is terminal **or** $t - t_{start} = N$
    Collect set of partial trajectories $\mathcal{D}$ on current policy π
    Estimate Advantages $\hat{A}_t^\pi = \sigma_t + (\gamma\lambda)\sigma_{t+1} + \cdots + (\gamma\lambda)^{N-t-1}\sigma_{N-1}$, where $\sigma_t = r_t + \gamma V(\phi_{t+1}) - V(\phi_t)$
    $\theta \leftarrow \text{argmax}_\theta \mathcal{L}_\theta^{CLIP}(\theta)$
    where $\mathcal{L}_\theta^{CLIP}(\theta) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\left[min(r_t(\theta)\hat{A}_t^\pi, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^\pi)\right]\right]$
    if ϕ_t is terminal **then**
      $e \leftarrow e + 1$
    end if
  until $e > e_{max}$
end for