The Physical Basis of Prediction: World Model Formation in Neural Organoids via an LLM-Generated Curriculum

Brennen Hill
Department of Computer Science
University of Wisconsin-Madison
Madison, WI 53706
bahill4@wisc.edu

Abstract

As the complexity of artificial agents increases, the design of environments that can effectively shape their behavior and capabilities has become a critical research frontier. We propose a framework that extends this principle to a novel class of agents: biological neural networks in the form of neural organoids. This paper introduces three scalable, closed-loop virtual environments designed to train organoid-based biological agents and probe the underlying mechanisms of learning, such as long-term potentiation (LTP) and long-term depression (LTD). We detail the design of three distinct task environments with increasing complexity: (1) a conditional avoidance task, (2) a one-dimensional predator-prey scenario, and (3) a replication of the classic Pong game. For each environment, we formalize the state and action spaces, the sensory encoding and motor decoding mechanisms, and the feedback protocols based on predictable (reward) and unpredictable (punishment) stimulation. Furthermore, we propose a novel meta-learning approach where a Large Language Model (LLM) is used to automate the generation and optimization of experimental protocols, scaling the process of environment and curriculum design. Finally, we outline a multi-modal approach for evaluating learning by measuring synaptic plasticity at electrophysiological, cellular, and molecular levels. This work bridges the gap between computational neuroscience and agent-based AI, offering a unique platform for studying embodiment, learning, and intelligence in a controlled biological substrate. Organoid Intelligence, Environment Scaling, LLM-Automated Design, Reinforcement Learning, Synaptic Plasticity, Embodied Biological Agents, Multi-Agent Systems, Virtual Environments

1 Introduction

The parallel scaling of model size, dataset size, and computation has yielded remarkable emergent capabilities in Large Language Models (LLMs) [Brown et al., 2020]. A crucial, yet comparatively underexplored, dimension for advancing agent intelligence is the scaling of environments. Environments provide the essential data for agents to acquire adaptive behaviors through interaction, moving beyond static imitation learning towards true end-to-end autonomy [Sutton and Barto, 2018]. The richness, diversity, and compositional structure of an environment directly shape an agent's ability to reason, plan, and generalize.

This paper explores these principles in the context of a novel agent substrate: living neural organoids. Recent advances in organoid technology [Lancaster et al., 2013] and brain-computer interfaces have enabled the creation of Organoid Intelligence (OI) systems, where 3D neural cultures are embodied

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Scaling Environments for Agents (SEA).

within simulated environments [Smirnova et al., 2023, Kagan et al., 2022]. These biological agents offer a unique opportunity to study the physical mechanisms of learning and memory, such as synaptic plasticity, in a human-derived neural system.

Here, we present a framework for designing, scaling, and evaluating simplified virtual environments for training organoid-based agents. We contribute three distinct experimental designs that form a curriculum of increasing complexity, focusing on fundamental agent capabilities like state avoidance, goal seeking, and dynamic interception. We formalize the environment infrastructure, including task formulation, action-space design, and agent integration via closed-loop electrophysiological stimulation and recording. Crucially, we also propose leveraging an LLM to automate the design of these experimental protocols, enabling a new paradigm for scaling research in this domain. By grounding agent learning in measurable biological changes like Long-Term Potentiation (LTP) and Long-Term Depression (LTD), our work offers a novel perspective on agent evaluation, moving beyond simple task performance to the underlying adaptation of the agent's neural architecture.

2 Related work

The concept of interfacing living neural tissue with artificial environments to study learning has a rich history. Early work with dissociated 2D neuronal cultures on multi-electrode arrays (MEAs) established the principle of in vitro embodiment, where neurally controlled animats learned to navigate a simulated world within a closed-loop system [DeMarse et al., 2001]. This foundational research demonstrated that biological neural networks could exhibit goal-directed behavior when provided with structured sensory feedback and the means to act upon their environment.

The transition from 2D cultures to 3D neural organoids represents a significant increase in biological fidelity, offering a model that more closely recapitulates the architecture, cellular diversity, and developmental programs of the human brain [Lancaster et al., 2013]. This complexity makes them a more powerful substrate for investigating network-level phenomena relevant to agent learning.

The free-engery principle [Friston, 2010] shows that predictable stimuli function as a reward (reinforcing actions that made the environment more predictable), while unpredictable, high-entropy stimuli act as a punishment. This demonstrated that goal-directed learning could be induced by structuring sensory feedback to align with the network's intrinsic drive to minimize surprise.

Furthermore, we incorporate a novel methodological approach by using an LLM to automate the design of experimental protocols. This aligns with an emerging trend of using AI for autonomous scientific discovery, where models have successfully optimized complex tasks in chemistry and materials science [Boiko et al., 2023], suggesting their immense potential to accelerate and scale research in computational neuroscience and agent training.

3 Methods

3.1 Multi-electrode array

All proposed experiments leverage a multi-electrode array (MEA) platform, capable of concurrent stimulation and recording from a biological culture. We assume a setup with four distinct electrode groups (A, B, C, D), making this setup simple to recreate. Each electrode group can contain multiple electrodes. Groups A and B are designated for recording neural activity (motor output), while groups C and D are used for stimulation (sensory input).

3.2 Biological substrates

While this paper focuses on neural organoids due to their structural complexity mimicking the human brain, the proposed experimental framework is fundamentally substrate-agnostic. These environments can be readily adapted for other biological preparations, including neural spheroids, 2D neuronal cultures, brain-on-a-chip platforms, or even non-neural tissue. The core requirement is a biological system that exhibits plasticity and can be integrated into a closed-loop system via an MEA for concurrent stimulation and recording. This flexibility allows for comparative studies across different levels of biological organization and complexity.

3.3 Agent feedback: Implementing reinforcement via the free-energy principle

The learning framework is predicated on principles of reinforcement learning (RL), where agent behavior is shaped by feedback from the environment. We operationalize this feedback through the lens of the free-energy principle [Friston, 2010], which posits that biological systems act to minimize surprise or prediction error. In this context, feedback signals are designed to be either predictable (low surprise) or unpredictable (high surprise), serving as proxies for positive and negative reinforcement, respectively. This mechanism provides the necessary error signals to drive synaptic plasticity, the biological substrate of learning.

Reward (positive reinforcement). A reward signal, corresponding to a positive outcome (e.g., capturing prey, intercepting the ball), is delivered as a predictable, low-entropy electrical stimulus to every electrode. A concrete implementation involves delivering a consistent, low-frequency sine wave across all stimulation electrodes. This predictable sensory input minimizes surprise and serves as a positive reward. Alternatively, a dose of dopamine can be delivered to the agent via UV light-controlled uncaging, which plays a role in inducing long-term potentiation [Schultz et al., 1997].

Punishment (negative reinforcement). Conversely, a punishment signal, corresponding to a negative outcome (e.g., entering an aversive zone, missing the ball), is delivered as an unpredictable, high-entropy stimulus. This is implemented by applying a white-noise electrical signal, characterized by random amplitude and frequency, to a randomly selected subset of stimulation electrodes. This chaotic and surprising sensory input functions as an aversive signal or punishment, thereby acting as a powerful error signal. The objective is to drive the network to alter its policy to avoid states and actions that lead to such high-surprise outcomes.

Together, this bipolar feedback scheme allows the agent to construct an internal model of its environment. By selectively reinforcing successful action sequences with predictable rewards and suppressing unsuccessful ones with unpredictable punishments, the framework directly translates the abstract principles of reinforcement learning into tangible biophysical changes within the neural network.

Encoding state. Information about the world state, such as the location of the organoid, can be encoded by stimulating a specific electrode. If there are 8 electrodes in a group, each can correspond to a different location. If there is only 1 electrode available, the location can be encoded by the frequency of the stimulation.

3.4 Benchmarks and evaluation: Measuring learning as plasticity

A unique advantage of using a biological agent is the ability to move beyond purely behavioral metrics and directly measure learning as a physical reorganization of the neural substrate. Our framework proposes a multi-scale evaluation strategy to quantify synaptic plasticity, correlating improvements in task performance with underlying biological changes. At the functional level, we can probe network connectivity and synaptic efficacy by measuring field excitatory postsynaptic potentials (fEPSPs) before and after training epochs. By delivering a test pulse to a stimulation electrode and recording the evoked potential in a downstream population, we can measure the fEPSP slope as a proxy for synaptic strength. A sustained increase in this slope post-training serves as a canonical indicator of long-term potentiation (LTP), while a decrease signifies long-term depression (LTD), providing a direct link between the agent's experience and Hebbian plasticity [Bliss and Lomo, 1973, Malenka and Bear, 2004].

To gain insight into how neural populations encode task variables, we can employ optical imaging at cellular resolution. For organoids expressing Genetically Encoded Calcium Indicators (GECIs) like GCaMP [Chen et al., 2013], two-photon microscopy allows for longitudinal tracking of the activity of hundreds of individual neurons. This approach enables us to observe how network dynamics adapt over the course of learning, identifying the emergence of stable neural ensembles that may represent specific game states, actions, or environmental features. Changes in the synchrony, frequency, and spatial patterns of these calcium transients can elucidate how the agent's internal model of the environment is formed and refined through interaction.

Finally, at the conclusion of an experiment, we can perform post-hoc molecular analyses to visualize the structural and chemical correlates of the observed functional changes. Immunohistochemistry can be used to probe the molecular machinery of plasticity. For instance, staining for the trafficking of

AMPA and NMDA receptor subunits can reveal changes in synaptic receptor density, a key mechanism underlying the expression of LTP and LTD [Malenka and Bear, 2004]. Furthermore, using antibodies against phosphorylated proteins that are critical for plasticity, such as pCaMKII or pGluA1, provides a molecular snapshot of synapses that were recently potentiated, thereby linking task-relevant activity to specific structural changes [Heo et al., 2023]. By integrating these electrophysiological, optical, and molecular benchmarks, we can construct a comprehensive picture of learning that bridges behavioral performance with the fundamental principles of neural adaptation.

4 Environments

4.1 Environment 1: Conditional avoidance

This initial experiment models a classic aversive conditioning task where the biological agent learns to associate a spatial region with negative stimuli and actively avoid it. This environment is inspired by similiar psychological experiments on mice [Miller, 1948]. A schematic of the environment is shown in Figure 1.

4.1.1 Protocol description

1. **State Representation and Sensory Encoding:** The agent's location within an 8-position one-dimensional grid is conveyed via patterned electrical stimulation. Each position (1-8) maps to a unique spatio-temporal stimulation pattern delivered across electrode groups C and D. This provides the organoid with proprioceptive information about its virtual embodiment. Positions 1-5 constitute a safe zone, while positions 6-8 form an aversive zone.

2. **Action Decoding and Execution:** The agent's intended movement is decoded by comparing the aggregate spike counts recorded from groups A and B over a discrete time window T. If the spike count from group A exceeds that of group B, the agent's virtual position is decremented (move left). Conversely, if group B's activity is higher, the position is incremented (move right). This mechanism translates differential network activity into a binary action space.

3. **Feedback and State Transition:** The environment provides feedback based on the agent's position. If the agent enters the aversive zone (positions 6-8), it receives an unpredictable, high-entropy electrical stimulus (punishment). The intensity of this stimulus is scaled with the depth of incursion (position $8 > 7 > 6$), providing a gradient for learning. Conversely, remaining in the safe zone for a sustained period of Z timesteps triggers a predictable reward.

4.1.2 Environmental scaling

- **Dimensionality:** The task can be scaled from a 1D line to a 2D grid or a 3D volume. In a 2D environment, the agent's state (x, y) could be encoded using separate electrode groups for each axis (e.g., Group C for x-coordinate, Group D for y-coordinate). The action space must expand to four actions (up, down, left, right), requiring a more complex motor decoding scheme, such as using four distinct recording sites or interpreting different spatio-temporal firing patterns from the existing two sites. A pseudocode implementation for this 2D extension is provided in Listing 5.
- **Zone Complexity:** The binary safe/aversive zone can be replaced with more complex spatial goals. For instance, the agent could be rewarded for staying on a narrow, winding path or for navigating a simple maze defined by aversive boundaries. This tests the organoid's ability to learn and represent more intricate spatial relationships. This requires replacing the simple boundary check with a function that evaluates the agent's position against a predefined geometric structure (see Listing 6).
- **Dynamic Environments:** To probe adaptability, the environment can be made non-stationary. The safe zone's boundary could slowly shift over the course of an experiment, or the exit of a maze could change location after a block of successful trials. This forces the agent to extinguish a previously learned policy and acquire a new one, providing a direct assay for cognitive flexibility (see Listing 7).

```
Initialize environment and agent interface
initialize_platform(groups_A, B, C, D)
```

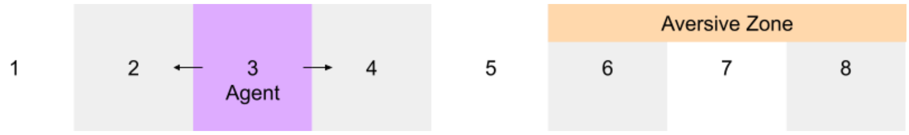

Figure 1: Diagram of the Conditional Avoidance environment. The agent, a neural organoid, can move left or right, and will receive negative feedback for entering the aversive zone.

```
agent_position = random.choice(columns_1_to_5)
safe_zone_timer = 0

Main experimental loop
while experiment_running:
# 1. State representation (sensory input)
encode_position_as_stimulus(groups_C_D, agent_position)

# 2. Action decoding (motor output)
spikes_A = record_spikes(group_A, window=T_ms)
spikes_B = record_spikes(group_B, window=T_ms)

if spikes_A > spikes_B:
    agent_position = max(1, agent_position - 1) # Move left
elif spikes_B > spikes_A:
    agent_position = min(8, agent_position + 1) # Move right

# 3. Feedback and state transition
if agent_position in columns_1_to_5: # In safe zone
    safe_zone_timer += 1
    if safe_zone_timer >= Z:
        deliver_reward(groups_C_D) # Predictable stimulus
        safe_zone_timer = 0
else: # In aversive zone
    safe_zone_timer = 0
    punishment_amplitude = calculate_amplitude(agent_position)
    deliver_punishment(groups_C_D, amplitude=punishment_amplitude)
```

Listing 1: Pseudocode for conditional avoidance task

4.2 Environment 2: Goal-seeking in a predator-prey scenario

This task increases complexity by requiring the agent (predator) to actively seek a goal (prey) rather than merely avoiding a negative state. This environment simulates the selective pressures inherent in foraging, a ubiquitous evolutionary challenge that requires an organism to develop efficient strategies for locating and capturing resources [Stephens and Krebs, 1986]. The environment can be scaled from a stationary to a moving prey to test for adaptive tracking. A diagram is shown in Figure 2.

4.2.1 Protocol description

1. **State Representation and Sensory Encoding:** The state is defined by the positions of both the predator and the prey on separate 8-position grids. The agent receives this information via two distinct sensory channels: the prey's location is encoded by stimulating a specific electrode in group C, while the agent's own (predator) location is encoded via group D. This separation provides clear exteroceptive (prey location) and proprioceptive (predator location) data streams.

2. **Action Decoding and Execution:** The motor decoding mechanism is identical to the previous environment, where differential activity between recording groups A and B determines left or right movement for the predator.

3. **Feedback and State Transition:** Reward is delivered when the predator's position matches the prey's position. Upon a successful capture, the prey respawns at a new random location. Punishment, in the form of an unpredictable stimulus, is delivered if the agent fails to capture the prey within a time limit of Z timesteps.

4.2.2 Environmental scaling

- **Dimensionality and Search Strategy:** Scaling to a 2D or 3D world transforms the task from simple directional movement to a genuine search problem. The agent must learn an efficient strategy (e.g., a patterned or random search) to locate the prey in a larger state space, placing greater demand on the network's ability to integrate sensory information over time. The sensory and motor schemes must be expanded to accommodate the higher dimensions, as shown in the pseudocode in Listing 8.
- **Introduction of a Predator:** A second mobile entity, a predator, can be added to the environment, creating a dual-objective task: approach prey while avoiding the predator. The predator's location would be encoded on a separate sensory channel (e.g., via a distinct stimulation frequency). Its behavior could range from a random walk to actively tracking the agent. Proximity to the predator would trigger a strong aversive stimulus and could reset the agent's position (see Listing 9).
- **Population Dynamics:** The environment can be populated with multiple prey, forcing the agent to develop a targeting strategy, or multiple predators, requiring more complex avoidance maneuvers.
- **Multi-Organoid Systems:** The framework allows for the creation of a simple ecosystem. One organoid could control the prey and another the predator, allowing their behaviors to co-evolve in a competitive dynamic. This can be extended to a linear food chain model (Organoid A is prey to B, B is prey to C) or a cyclical model to study the emergence of hierarchical strategies in a multi-agent biological system. A pseudocode implementation for a two-organoid system is available in Listing 10.

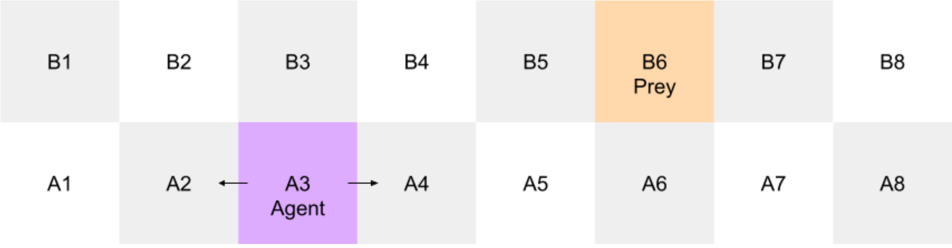

Figure 2: Diagram of the 1D Predator-prey environment. The agent, a neural organoid, can move left or right, and will be rewarded for occupying the same column as the prey.

```
Initialize environment
predator_pos = random.choice(A1_to_A8)
prey_pos = random.choice(B1_to_B8)
time_since_reward = 0

Main experimental loop
while experiment_running:
# Optional: Update prey position for dynamic version
# prey_pos = move_prey_procedurally(prey_pos)

# 1. State representation
encode_stimulus(group_C, position=prey_pos)
encode_stimulus(group_D, position=predator_pos)

# 2. Action decoding
spikes_A = record_spikes(group_A, window=T_ms)
spikes_B = record_spikes(group_B, window=T_ms)

if spikes_A > spikes_B:
    predator_pos = max(1, predator_pos - 1)
elif spikes_B > spikes_A:
    predator_pos = min(8, predator_pos + 1)

# 3. Feedback and state transition
```

```
if predator_pos == prey_pos:
    deliver_reward(groups_C_D)
    prey_pos = random.choice(B1_to_B8) # Respawn prey
    time_since_reward = 0
else:
    time_since_reward += 1

if time_since_reward >= Z:
    deliver_punishment(groups_C_D)
    prey_pos = random.choice(B1_to_B8) # Respawn prey
    time_since_reward = 0
```

Listing 2: Pseudocode for predator-prey task

4.3 Environment 3: Dynamic interception task (Pong)

This experiment replicates the dynamic, real-time control task from [Kagan et al., 2022], providing a benchmark for goal-directed behavior in a more complex, continuous state space. The experimental layout is depicted in Figure 3.

4.3.1 Protocol description

1. **Game State Update:** The simulation first updates the ball's position based on its current velocity vector, modeling basic physics within the 2D game world.

2. **State Representation and Sensory Encoding:** The continuous 2D position of the ball is conveyed to the organoid through a multi-modal encoding scheme. The ball's horizontal location (X-position) is spatially encoded, where discrete positions maps to specific electrodes across stimulation groups C and D. If each group consists of multiple electrodes, the position will be more precise. Concurrently, the ball's vertical distance from the paddle (Y-position) is rate-coded, with stimulation frequency being inversely proportional to distance (e.g., 40 Hz when closest, 4 Hz when farthest). This complex sensory input requires the network to integrate two distinct data streams.

3. **Action Decoding and Execution:** The agent controls its paddle's vertical movement. Higher aggregate spike rates from recording group A over a 10 ms window move the paddle up, while higher rates from group B move it down.

4. **Feedback:** A successful interception results in a reward, while a failure triggers an unpredictable punishment.

4.3.2 Environmental scaling

- **Generalization to Other Games:** The Pong paradigm can be adapted to other classic arcade games to probe different cognitive functions. For example, Breakout would require precise spatial targeting to aim the ball at specific bricks. Space Invaders would add the complexities of dodging projectiles while timing offensive actions (requiring a third shoot action in the decoded output). A simplified version of Pac-Man would test maze navigation, goal-seeking, and state-dependent behavior (i.e., switching from avoiding ghosts to hunting them). A pseudocode implementation for adapting the task to Breakout is provided in Listing 11.
- **Competitive Multi-Organoid Systems:** Pong is an ideal testbed for multi-agent competition. Two separate organoid-MEA systems can be linked to the same game instance, each controlling one of the paddles. This setup allows for the direct study of competitive co-adaptation, where each biological agent must learn and adapt its strategy in real-time response to the actions of the other. The evolution of rally lengths and win/loss rates would serve as powerful metrics for inter-agent learning (see Listing 12).

4.4 Automated protocol generation using a large language model

To scale the exploration of experimental parameters and curriculum design, we propose an automated framework where an LLM functions as a meta-controller. The LLM is applied as a tool to orchestrate the agent's training regimen rather than the agent itself. Manual design of experimental protocols is slow, limited by researcher intuition, and may not efficiently explore the vast parameter space.

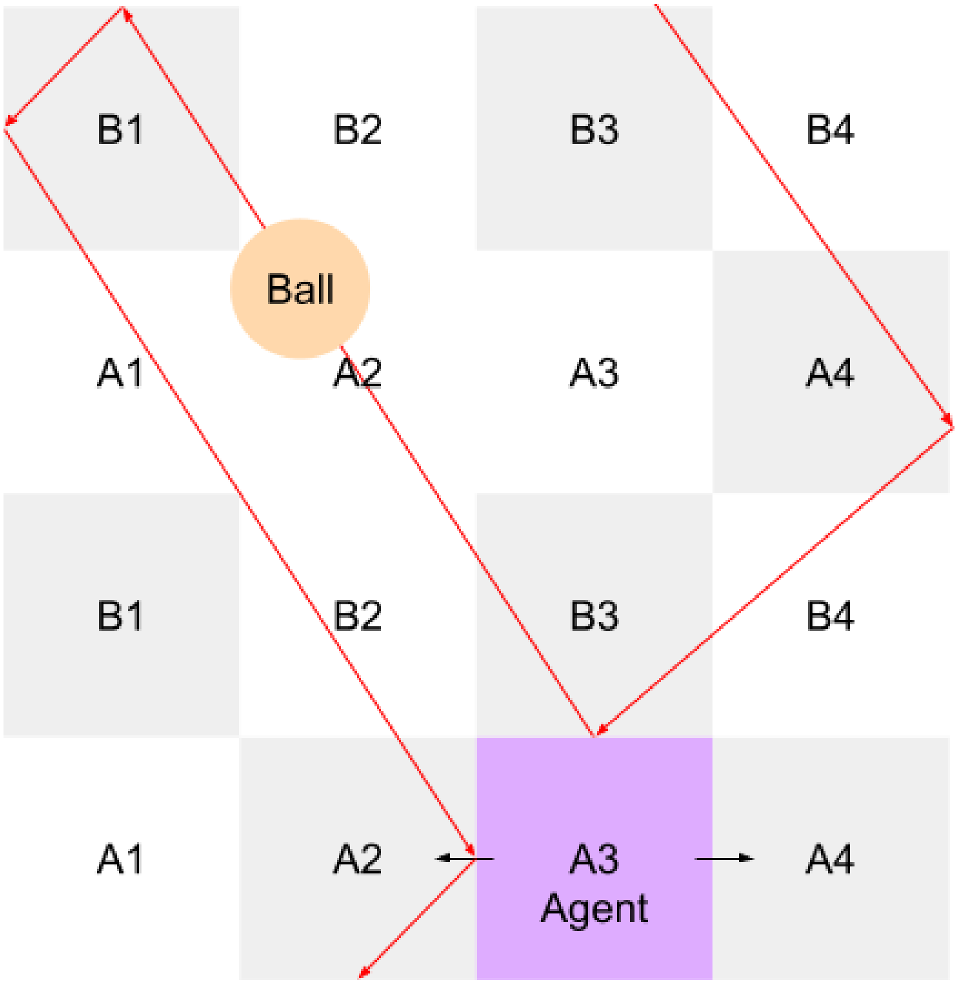

Figure 3: The Pong environment from. The organoid receives complex, multi-modal sensory input about the ball's position (rate-coded distance, spatially-coded location) and controls the paddle through differential neural activity.

Automating this process creates a high-throughput system for discovering optimal training strategies and environmental configurations. A pseudocode implementation of this loop is provided in Listing 4.

4.4.1 Framework description

The system operates in a closed loop, iteratively generating, executing, and refining experimental protocols. An empty dataset is created to store all information from each iteration, including generated scripts, execution outputs, performance metrics (e.g., capture rate), neurophysiological responses, and any errors.

1. **Prompt Formulation:** At the start of an iteration, a detailed prompt is formulated for the LLM via its API. This prompt includes: (1) the high-level experimental objective (e.g., maximize prey capture rate); (2) a description of the available API commands and their valid parameter ranges, covering electrical stimulation (e.g., shape from [bi-phasic, tri-phasic], amplitude from 0.1-20.0 µA, pulse_duration from 50-500 µs, pulse train frequency from 5-100 Hz), microfluidics (flow_rate), and UV uncaging (duration from 100-1000 ms); (3) the current state of the organoid, such as baseline firing rates; and (4) the historical dataset of past iterations, summarizing which protocols succeeded or failed.

2. **Protocol Generation:** The LLM generates a novel experimental protocol based on the prompt. This can be done in two primary modes: generating a JSON object with specific experimental

variables to be plugged into a template script, or generating an entire Python script from scratch that defines the experimental logic. We provide detailed examples of both approaches in Appendix A.3.

3. **Validation and Execution:** The generated protocol is programmatically validated. A parser checks for syntactic correctness and verifies that all specified parameters fall within safe, predefined ranges. If the protocol is invalid, it is logged to the dataset as erroneous, and the loop returns to the previous step. If valid, the script is executed on the MEA platform. This involves neurosphere stimulation, data collection, administering rewards or punishments, and updating the experiment state.

4. **Data Logging:** All data from the loop iteration are saved to the dataset. This includes the generated algorithm, the neurosphere's response, performance metrics, and the resulting experimental state. This comprehensive record is crucial for the refinement stage.

5. **Iterative Refinement:** Steps 1-4 are repeated for a desired batch size. After each batch, the collected dataset is used to evaluate and refine the LLM's performance. By identifying patterns in successes and failures, we can improve the LLM through two main avenues:

- **Prompt Engineering:** The base prompt is updated. This can involve adding successful examples from the dataset for few-shot prompting, incorporating chain-of-thought reasoning by asking the LLM to explain its choices, or adding explicit constraints based on failed experiments.
- **Fine-tuning:** For more significant adaptation, the model can be fine-tuned using the dataset. Successful prompt-protocol pairs serve as training data for supervised fine-tuning, teaching the model to replicate effective strategies. This creates a powerful feedback loop where the LLM becomes an increasingly competent experimental designer over time.

5 Discussion

The framework presented in this paper sits at the intersection of agent-based AI and neuroscience, and its implications warrant careful consideration.

5.1 Broader implications and future work

Treating an organoid as an agent, rather than a passive model, is a paradigm shift. It reframes questions from "What does this neural activity represent?" to "What is this agent trying to do?" This perspective naturally aligns with reinforcement learning principles and predictive coding theories like the Free Energy Principle, suggesting a common computational language for both biological and artificial intelligence.

Looking ahead, the logical progression is to scale the complexity of both the agent and the environment. This includes:

- **Environment Scaling:** Moving beyond simple 1D and 2D tasks to more complex, physics-based simulations or even multi-agent scenarios where organoids could interact with each other or with *in silico* agents.
- **Interface Scaling:** Increasing the communication bandwidth through high-density MEAs or integrating optogenetics for precise, cell-type-specific stimulation. This would create a richer observation space and a more expressive action space for the agent.
- **Sim2Real and Embodiment:** A major future direction is to bridge the *in vitro*-to-physical gap. By connecting these systems to simple robotic actuators and sensors, we could create truly embodied biological agents, providing a powerful platform for studying grounded cognition and tackling Sim2Real challenges from a completely new angle.

5.2 Limitations and challenges

Despite its promise, this approach faces significant hurdles. The biological substrate introduces challenges not present in *in silico* research. Organoids exhibit high variability, have limited lifespans, and their developmental trajectories are difficult to standardize, making reproducibility a key concern. The black box nature of the organoid is profound; while we can measure network-level plasticity, understanding the specific algorithms it implements remains an open question. Furthermore, the information bandwidth of current MEAs is extremely low, potentially constraining the complexity of learnable tasks.

6 Conclusion

This paper presents a framework for the design, scaling, and evaluation of virtual environments for a novel class of biological agent: the neural organoid. By structuring tasks as a curriculum of increasing complexity, from conditional avoidance to dynamic interception, we can systematically probe and induce learning. The proposed methods for sensory encoding, motor decoding, and feedback are grounded in established principles of neuroscience and agent-based learning.

Our key contributions are threefold: (1) we provide a concrete set of scalable environment designs tailored for inducing learning in biological neural networks; (2) we introduce a novel evaluation paradigm for agents, linking behavioral performance to direct measurement of synaptic plasticity; and (3) we propose a forward-looking, LLM-driven methodology for automating and scaling the design of training protocols, which has broad implications for both Organoid Intelligence and AI-driven scientific discovery. This work represents a critical step towards creating more sophisticated and adaptive biological agents, offering a powerful platform to explore the fundamental connections between environment, embodiment, and intelligence.

Acknowledgments and Disclosure of Funding

We thank FinalSpark for proposing the idea of LLM based automated research.

References

T V Bliss and T Lomo. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of physiology*, 232 (2):331–356, 1973.

Daniil A Boiko, Robert MacKnight, Gabe Kline, and Gabriel Gomes. Autonomous chemical research with a large language model. *Nature*, 624(7992):570–578, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Tsai-Wen Chen, Trevor J Wardill, Yi Sun, Stefan R Pulver, Sabine L Renninger, Amy Baohan, Eric R Schreiter, Rex A Kerr, Michael B Orger, Vivek Jayaraman, et al. Ultrasensitive fluorescent proteins for imaging neuronal activity. *Nature*, 499(7458):295–300, 2013.

Thomas B DeMarse, Daniel A Wagenaar, Andreas W Blau, and Steve M Potter. The neurally controlled animat: biological brains acting with simulated bodies. *Autonomous robots*, 11(3): 305–310, 2001.

Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2): 127–138, 2010.

Seok Heo, Taewook Kang, Alexei M Bygrave, Martin R Larsen, and Richard L Huganir. Experience-induced remodeling of the hippocampal post-synaptic proteome and phosphoproteome. *Molecular & Cellular Proteomics*, 22(11), 2023.

Fred D Jordan, Martin Kutter, Jean-Marc Comby, Flora Brozzi, and Ewelina Kurtys. Open and remotely accessible neuroplatform for research in wetware computing. *Frontiers in Artificial Intelligence*, 7, 2024. doi: 10.3389/frai.2024.1376042.

Brett J Kagan, Andy C Kitchen, Nhi T Tran, Bradyn J Parker, Anjali Bhat, Ben Rollo, Adeel Razi, and Karl J Friston. In vitro neurons learn and exhibit sentience when embodied in a simulated game-world. *Neuron*, 110(21):3552–3568, 2022.

Madeline A Lancaster, Magdalena Renner, Carol-Anne Martin, Daniel Wenzel, Louise S Bicknell, Matthew E Hurles, Tessa Homfray, Josef M Penninger, Andrew P Jackson, and Juergen A Knoblich. Cerebral organoids model human brain development and microcephaly. *Nature*, 501(7467):373–379, 2013.

Robert C Malenka and Mark F Bear. Ltp and ltd: an embarrassment of riches. *Neuron*, 44(1):5–21, 2004.

Neal E. Miller. Studies of fear as an acquirable drive: I. fear as motivation and fear-reduction as reinforcement in the learning of new responses. *Journal of Experimental Psychology*, 38(1): 89–101, 1948.

Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

Lena Smirnova, Brian S Caffo, David H Gracias, Qi Huang, Itzy E Morales Pantoja, Bohao Tang, Donald J Zack, Cynthia A Berlinicke, J Lomax Boyd, Timothy D Harris, et al. Organoid intelligence (oi): the new frontier in biocomputing and intelligence-in-a-dish. *Frontiers in Science*, 1:1017235, 2023.

David W. Stephens and John R. Krebs. *Foraging theory*. Princeton University Press, 1986.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, second edition, 2018.

A Technical appendices and supplementary material

A.1 Additional pseudocode for initial environments and automation

This section provides the base pseudocode for the Pong environment and the LLM-driven automation framework described in the main text.

```
Initialize game state and MEA interface
game = PongGame()

Main game loop
while game.is_active:
# 1. Update game state
game.update_ball_position()
ball_x, ball_y = game.get_ball_position()

# 2. State representation
encode_y_pos_rate_coded(groups_C_D, ball_y)
encode_x_pos_spatial(groups_C_D, ball_x)

# 3. Action decoding
spikes_A = record_spikes(group_A, window=10_ms)
spikes_B = record_spikes(group_B, window=10_ms)

if spikes_A > spikes_B:
    game.move_paddle_up()
elif spikes_B > spikes_A:
    game.move_paddle_down()

# 4. Feedback
if game.check_collision():
    deliver_reward_predictable(groups_C_D)
    game.ball_bounce()
elif game.check_miss():
    deliver_punishment_unpredictable(groups_C_D)
    game.reset_ball()
```

Listing 3: Pseudocode for pong task

```
Initialize dataset and load base prompt template
dataset = initialize_database()
prompt_template = load_base_prompt("config/llm_prompt.txt")

Main automation loop
for iteration in range(NUM_ITERATIONS):
# 1. Formulate prompt with historical data
current_organoid_state = get_organoid_baseline()
prompt = formulate_prompt(prompt_template, dataset,
    current_organoid_state)

# 2. LLM generates a new experimental script
generated_script = call_llm_api(prompt)

# 3. Validate the generated script for safety and syntax
is_valid, error_msg = validate_protocol_script(generated_script)

# 4. Execute or log error
if is_valid:
    # Run the experiment on the MEA platform
    results = execute_script_on_platform(generated_script)

    # Store the script and its outcomes in the dataset
    store_results(dataset, script=generated_script, results=results)
else:
    # Log the invalid script and the reason for failure
    store_error(dataset, script=generated_script, error=error_msg)

# 5. Periodically refine the LLM's instructions
if (iteration + 1) % BATCH_SIZE == 0:
    prompt_template = refine_prompt_based_on_results(dataset)
    # Optional: Trigger fine-tuning if sufficient data is collected
    # if len(dataset) > FINETUNE_THRESHOLD:
    #     launch_finetuning_job(dataset)
```

Listing 4: Pseudocode for LLM-driven experiment automation

A.2 Pseudocode for scaled environments

This section provides pseudocode implementations for the scaling variations. The algorithms are grouped by the base environment from which they are extended, providing concrete examples of how task complexity can be systematically increased.

A.2.1 Scaling the conditional avoidance environment

The algorithms below correspond to the scaling variations for the Conditional Avoidance task, including extensions to 2D space, maze navigation, and dynamic (non-stationary) conditions.

```
Initialize a 10x10 grid environment
GRID_SIZE = 10
agent_pos = (random.randint(0, GRID_SIZE-1), random.randint(0, 4)) #
    Start in safe half
aversive_zone_x = range(GRID_SIZE // 2, GRID_SIZE) # Right half is
    aversive

Assume 4 recording sites (A_up, A_down, B_left, B_right)
initialize_platform(groups_A_up, A_down, B_left, B_right, C, D)

while experiment_running:
# 1. State representation (2D)
encode_position_stimulus(group_C, axis='x', value=agent_pos[0])
encode_position_stimulus(group_D, axis='y', value=agent_pos[1])
```

```
# 2. Action decoding (4-way movement)
spikes_up = record_spikes(group_A_up, window=T_ms)
spikes_down = record_spikes(group_A_down, window=T_ms)
spikes_left = record_spikes(group_B_left, window=T_ms)
spikes_right = record_spikes(group_B_right, window=T_ms)

action = determine_dominant_action(spikes_up, spikes_down, spikes_left
    , spikes_right)

# Update position based on decoded action
x, y = agent_pos
if action == ’UP’:    y = min(GRID_SIZE - 1, y + 1)
if action == ’DOWN’:  y = max(0, y - 1)
if action == ’LEFT’:  x = max(0, x - 1)
if action == ’RIGHT’: x = min(GRID_SIZE - 1, x + 1)
agent_pos = (x, y)

# 3. Feedback based on 2D zone
if agent_pos[0] in aversive_zone_x:
    punishment_amplitude = calculate_amplitude(agent_pos[0])
    deliver_punishment(groups_C_D, amplitude=punishment_amplitude)
else:
    deliver_reward(groups_C_D) # Simplified: reward for being in safe
        zone
```

Listing 5: Pseudocode for 2D conditional avoidance

```
Define a maze as a grid where 1=path, 0=wall
maze_layout = [
[1, 1, 1, 0, 1],
[0, 1, 0, 0, 1],
[1, 1, 1, 1, 1],
[1, 0, 1, 0, 1],
[1, 1, 1, 0, 1]
]
start_pos, goal_pos = (0, 0), (4, 4)
agent_pos = start_pos

while experiment_running:
# State and Action decoding remain similar to 2D avoidance
encode_position_stimulus(group_C, axis=’x’, value=agent_pos[0])
encode_position_stimulus(group_D, axis=’y’, value=agent_pos[1])

action = decode_4way_movement(groups_A_B)
next_pos = calculate_next_pos(agent_pos, action)

# 3. Feedback and state transition based on maze structure
x, y = next_pos
if maze_layout[y][x] == 1: # Valid path
    agent_pos = next_pos
    if agent_pos == goal_pos:
        deliver_reward(groups_C_D)
        agent_pos = start_pos # Reset to start
else: # Hit a wall
    deliver_punishment(groups_C_D)
    # Agent does not move if it tries to enter a wall
```

Listing 6: Pseudocode for maze navigation task

```
Initialize environment with a moving boundary
agent_position = 4
safe_zone_boundary = 5 # Initially, positions 1-5 are safe
boundary_shift_interval = 100 # Timesteps before boundary moves
timestep = 0
```

```
while experiment_running:
# 1. State representation & 2. Action decoding (same as 1D)
encode_position_as_stimulus(groups_C_D, agent_position)
agent_position = update_position_from_spikes(groups_A_B,
    agent_position)

# 3. Feedback based on the DYNAMIC boundary
if agent_position <= safe_zone_boundary:
    deliver_reward(groups_C_D)
else:
    deliver_punishment(groups_C_D)

# 4. Update the environment itself
timestep += 1
if timestep % boundary_shift_interval == 0:
    # Shift the boundary occasionally, making the task harder
    if random.choice(range(0, 1 )) == 1:
        safe_zone_boundary = max(1, safe_zone_boundary - 1)
    else:
        safe_zone_boundary = min(safe_zone_boundary + 1, 8)
```

Listing 7: Pseudocode for dynamic avoidance task

A.2.2 Scaling the predator-prey environment

These examples illustrate extensions to the predator-prey scenario. This includes increasing dimensionality, introducing an adversary, and creating a multi-agent competitive system.

```
Initialize a 10x10 grid environment
GRID_SIZE = 10
predator_pos = (random.randint(0, 9), random.randint(0, 9))
prey_pos = (random.randint(0, 9), random.randint(0, 9))

while experiment_running:
# 1. State representation (2D)
# Encode prey position using frequency on group C
encode_stimulus(group_C, axis=’x’, value=prey_pos[0])
encode_stimulus(group_C, axis=’y’, value=prey_pos[1], use_freq_mod=
    True)
# Encode predator position on group D
encode_stimulus(group_D, axis=’x’, value=predator_pos[0])
encode_stimulus(group_D, axis=’y’, value=predator_pos[1], use_freq_mod
    =True)

# 2. Action decoding (4-way movement)
action = decode_4way_movement(groups_A_B)
predator_pos = update_2d_position(predator_pos, action, GRID_SIZE)

# 3. Feedback for 2D capture
if predator_pos == prey_pos:
    deliver_reward(groups_C_D)
    prey_pos = (random.randint(0, 9), random.randint(0, 9))
```

Listing 8: Pseudocode for 2D predator-prey task

```
Add a third entity, the adversary
predator_pos = 5
prey_pos = 2
adversary_pos = 8
DANGER_RADIUS = 2

while experiment_running:
# Adversary moves with a simple policy (e.g., towards predator)
```

```
adversary_pos = move_adversary(adversary_pos, predator_pos)

# 1. State representation (includes adversary location)
encode_stimulus(group_C, position=prey_pos, pattern=’prey’)
encode_stimulus(group_D, position=predator_pos, pattern=’self’)
# Use different frequency/amplitude to encode adversary
encode_stimulus(group_C, position=adversary_pos, pattern=’adversary’)

# 2. Action decoding (same as 1D)
predator_pos = update_position_from_spikes(groups_A_B, predator_pos)

# 3. Feedback with dual objective
if abs(predator_pos - adversary_pos) < DANGER_RADIUS:
    deliver_punishment(groups_C_D, amplitude=’high’)
    predator_pos = random.choice(range(1,9)) # Reset on capture
elif predator_pos == prey_pos:
    deliver_reward(groups_C_D)
    prey_pos = random.choice(range(1,9))
```

Listing 9: Pseudocode for predator-prey with an adversary

```
Initialize two separate MEA platforms
organoid_predator = MEA_Platform(id=1)
organoid_prey = MEA_Platform(id=2)

Initialize shared game state
predator_pos = 5
prey_pos = 3

while experiment_running:
# PREDATOR ORGANOID’S TURN
# 1a. Send state to predator organoid
organoid_predator.encode_state(predator_pos, prey_pos)
# 2a. Get action from predator organoid
predator_action = organoid_predator.decode_action()
predator_pos = update_position(predator_pos, predator_action)

# PREY ORGANOID’S TURN
# 1b. Send state to prey organoid
organoid_prey.encode_state(prey_pos, predator_pos)
# 2b. Get action from prey organoid
prey_action = organoid_prey.decode_action()
prey_pos = update_position(prey_pos, prey_action)

# 3. Deliver feedback to both based on outcome
if predator_pos == prey_pos:
    organoid_predator.deliver_reward()
    organoid_prey.deliver_punishment()
    # Reset positions
    predator_pos, prey_pos = 5, 3
```

Listing 10: Pseudocode for multi-organoid predator-prey

A.2.3 Scaling the dynamic interception (Pong) environment

The following pseudocode shows how the principles of the Pong task can be generalized to other classic games like Breakout or adapted for multi-agent competition.

```
Initialize Breakout game state
game = BreakoutGame() # Includes paddle, ball, and bricks

while game.is_active:
# 1. Update game state (ball movement, etc.)
game.update_ball_position()
```

```
# 2. State representation
# Same as Pong: encode ball x (spatial) and y (rate-coded)
encode_ball_state(groups_C_D, game.get_ball_position())
# Could add info about brick layout (e.g., density in ball’s path)

# 3. Action decoding (paddle movement)
spikes_A = record_spikes(group_A, window=10_ms)
spikes_B = record_spikes(group_B, window=10_ms)
if spikes_A > spikes_B:
    game.move_paddle_left()
elif spikes_B > spikes_A:
    game.move_paddle_right()

# 4. Feedback
collision_type = game.check_collisions()
if collision_type == ’BRICK’:
    deliver_reward_predictable(groups_C_D)
    game.remove_brick()
    game.ball_bounce()
elif collision_type == ’PADDLE’:
    # Neutral or small reward for just keeping ball in play
    game.ball_bounce()
elif collision_type == ’MISS’:
    deliver_punishment_unpredictable(groups_C_D)
    game.reset_ball()
```

Listing 11: Pseudocode for breakout task

```
Initialize two organoid interfaces and one shared game
organoid_1 = MEA_Platform(id=1)
organoid_2 = MEA_Platform(id=2)
game = PongGame(two_player=True)

while game.is_active:
# 1. Update game state
game.update_ball_position()
ball_pos = game.get_ball_position()

# 2. Send state to both organoids
# Each organoid receives the same information
organoid_1.encode_ball_state(ball_pos)
organoid_2.encode_ball_state(ball_pos)

# 3. Decode actions from both organoids simultaneously
action_1 = organoid_1.decode_paddle_movement()
action_2 = organoid_2.decode_paddle_movement()
game.move_paddle(player=1, action=action_1)
game.move_paddle(player=2, action=action_2)

# 4. Check for outcomes and deliver feedback independently
if game.check_paddle1_miss():
    organoid_1.deliver_punishment()
    organoid_2.deliver_reward() # Reward for opponent’s miss
    game.reset_ball(server=2)
elif game.check_paddle2_miss():
    organoid_2.deliver_punishment()
    organoid_1.deliver_reward()
    game.reset_ball(server=1)
elif game.check_paddle_hit():
    game.ball_bounce()
    # Optional: deliver small reward to hitting player
```

Listing 12: Pseudocode for two-organoid competitive pong

A.3 LLM prompting examples

This appendix provides detailed examples of the two primary modes for using an LLM to generate experimental protocols: structured JSON output for parameter tuning and full Python script generation for implementing novel logic and curriculum design.

A.3.1 Example 1: JSON-based parameter optimization

This approach is ideal for systematic exploration of a known parameter space. By constraining the LLM to output a JSON object, we ensure the protocol is always syntactically correct and can be easily parsed and validated before execution. This method is particularly useful for optimizing feedback mechanisms, such as reward and punishment, by exploring different stimulation modalities and their specific parameters.

The following prompt (Listing 13) asks the LLM to propose a new set of parameters for the predator-prey task, taking into account historical data and a new hypothesis. It must decide between an electrical reward and a chemical (dopamine uncaging) reward, and then specify the nested parameters for its choice.

Listing 13: Prompt for LLM to generate JSON parameters, including reward modality.

```
You are an AI assistant optimizing a neuroscience experiment.
Goal: Increase the prey capture rate for a biological agent. Current
    rate is 25%.

Historical Data:
- Run 1: Electrical reward (20Hz, 2uA) -> 22% capture rate.
- Run 2: Electrical reward (40Hz, 2uA) -> 24% capture rate.
- Run 3: Dopamine uncaging reward (500ms duration) -> 25% capture rate
    , but learning is slow.

Hypothesis: A stronger, more salient electrical reward might be more
    effective than the previous attempts. Let's try a tri-phasic pulse
     to ensure charge balance and a higher amplitude.

Propose a new set of parameters. Your output must be a valid JSON
    object.
Parameter constraints:
- reward_modality: one of ["electrical", "dopamine_uncaging"]
- electrical_params.shape: one of ["bi-phasic", "tri-phasic"]
- electrical_params.amplitude_uA: float, range [0.1, 20.0]
- electrical_params.pulse_duration_us: int, range [50, 500]
- dopamine_params.uncaging_duration_ms: int, range [100, 1000]
```

An example of an LLM response from our trials is shown in Listing 14. The model correctly interprets the hypothesis, selects the electrical modality, and populates the nested parameters accordingly, including the requested tri-phasic shape and a significantly higher amplitude_uA. This structured output can be directly loaded by the experimental control software to configure the next run.

Listing 14: Example of a JSON output from the LLM with nested parameters.

```
{
  "reward_modality": "electrical",
  "electrical_params": {
    "shape": "tri-phasic",
    "amplitude_uA": 8.5,
    "pulse_duration_us": 150,
    "frequency_hz": 30
  },
  "dopamine_params": null,
  "punishment_params": {
    "shape": "bi-phasic",
    "amplitude_uA": 10.0,
    "pulse_duration_us": 200
  }
}
```

A.3.2 Example 2: Full script generation for curriculum design

This approach grants the LLM maximum flexibility to design novel experimental logic, making it suitable for tasks like curriculum design, where the protocol itself must change based on agent performance. By providing the LLM with a detailed API specification, it can compose complex sequences of operations that would be impossible to specify in a simple key-value format.

The prompt in Listing 15 outlines a more comprehensive API and asks the LLM to write a complete Python script for a block of trials. This includes defining the logic for state encoding, action decoding, feedback delivery, and, crucially, a terminal condition with a suggestion for the next curriculum step. This example uses FinalSpark's Neuroplatform [Jordan et al., 2024]

Listing 15: Prompt for LLM to generate a full Python script with a detailed API.

```
You are an AI assistant designing a Python script for the
    Neuroplatform.
Task: Write a script for one block of the predator-prey experiment.
Goal: Teach the agent to capture a stationary prey.

Use the provided NeuroplatformAPI. Key functions:
- api.encode_spatial(electrode_num): Stimulate one electrode for
    location.
- api.get_spike_counts(electrodes_A, electrodes_B): Returns (count_A,
    count_B).
- api.fire_pulse_train(electrodes, shape, amplitude_uA, pulse_us,
    freq_hz, duration_ms): Delivers electrical feedback.
- api.uncage_dopamine(duration_ms): Delivers chemical reward.

Write a Python script for a 10-trial block.
- Prey is stationary at position 6 (encoded on electrode 14).
- Predator starts at position 2 (encoded on electrode 2).
- Reward: A 500ms pulse train at 40Hz on all sensory electrodes.
- Punishment: A high-amplitude (15uA) single bi-phasic pulse.
At the end, print the capture rate and suggest a curriculum step. If
    capture rate > 70%, suggest making the prey move. Otherwise,
    suggest repeating.
```

The resulting script (Listing 16) is an example from our trials. It demonstrates the LLM's ability to correctly use the specified API. It correctly structures the trial loop, implements the state encoding and action decoding logic, and calls the complex fire_pulse_train function with the correct parameters for both reward and punishment. Finally, it calculates the performance metric and prints a context-aware suggestion, effectively automating a step in curriculum design.

Listing 16: Example of a Python script generated by the LLM using a detailed API.

```python
from neuroplatform_api import NeuroplatformAPI
import time

def run_predator_prey_block(api):
    num_trials = 10
    captures = 0

    # Define sensory and motor electrodes
    SENSORY_ELECTRODES = [10, 11, 12, 13, 14, 15, 16, 17]
    MOTOR_A_ELECTRODES = [0, 1]
    MOTOR_B_ELECTRODES = [8, 9]

    for trial in range(num_trials):
        predator_pos = 2
        prey_pos = 6

        for timestep in range(15): # Max 15 steps per trial
```

```
            api.encode_spatial(electrode_num=prey_pos + 8) # Prey on
                group D
            api.encode_spatial(electrode_num=predator_pos) # Predator
                on group C
            time.sleep(0.1)

            spikes_A, spikes_B = api.get_spike_counts(
                MOTOR_A_ELECTRODES, MOTOR_B_ELECTRODES)

            if spikes_A > spikes_B:
                predator_pos = max(1, predator_pos - 1)
            elif spikes_B > spikes_A:
                predator_pos = min(8, predator_pos + 1)

            if predator_pos == prey_pos:
                # Deliver complex electrical reward
                api.fire_pulse_train(
                    electrodes=SENSORY_ELECTRODES, shape='bi-phasic',
                    amplitude_uA=5.0, pulse_us=200,
                    freq_hz=40, duration_ms=500
                )
                captures += 1
                break

        if predator_pos != prey_pos: # Punishment if not captured
            api.fire_pulse_train(
                electrodes=SENSORY_ELECTRODES, shape='bi-phasic',
                amplitude_uA=15.0, pulse_us=200,
                freq_hz=0, duration_ms=1 # Single pulse
            )

    capture_rate = captures / num_trials
    print(f"Block␣finished.␣Capture␣rate:␣{capture_rate:.2f}")
    if capture_rate > 0.7:
        print("Suggestion␣for␣next␣block:␣Success.␣Make␣prey␣move␣
            predictably.")
    else:
        print("Suggestion␣for␣next␣block:␣Low␣performance.␣Repeat␣
            block.")

if __name__ == '__main__':
    platform_api = NeuroplatformAPI()
    run_predator_prey_block(platform_api)
```