Neuroweaver: Towards a Platform for Designing Translatable Intelligent Closed-loop Neuromodulation Systems

Parisa Sarikhani
Emory University
psarikh@emory.edu

Hao-Lun Hsu
Georgia Institute of Technology
hhsu61@gatech.edu

Joon Kyung Kim
University of California San Diego
jkkim@eng.ucsd.edu

Sean Kinzer
University of California San Diego
skinzer@eng.ucsd.edu

Edwin Mascarenhas
University of California San Diego
emascare@ucsd.edu

Hadi Esmaeilzadeh
University of California San Diego
hadi@eng.ucsd.edu

Mahmoudi Babak
Emory University and Georgia Institute of Technology
b.mahmoudi@emory.edu

Abstract

Closed-loop neuromodulation provides a powerful paradigm for the treatment of diseases, restoring function, and understanding the causal links between neural and behavioral processes, however the complexities of interacting with the nervous system create several challenges for designing optimal closed-loop neuromodulation control systems and translating them into clinical settings. Artificial Intelligence (AI) and Reinforcement Learning (RL) can be leveraged to design intelligent closed-loop neuromodulation (iCLON) systems that can autonomously learn and adapt neuromodulation control policies in clinical settings and bridge the translational gap between pre-clinical design and clinical deployment of neuromodulation therapies. We are developing an open-source AI platform, called Neuroweaver, to enable algorithm-software-hardware co-design and deployment of translatable iCLON systems. In this paper, we present the design elements of the Neuroweaver platform that are translatability of iCLON systems.

1 Introduction

Neuromodulation technologies such as electrical and optogenetic stimulation of neural systems are powerful tools for understanding the causal link between neural and behavioral processes and provide promising treatment options for neurological, psychiatric, and neurodegenerative disorders. In fact, deep brain stimulation (DBS) has become a standard treatment for movement disorders such as Parkinson's disease and essential tremor[1]. Most of the currently adopted clinical DBS approaches deliver the stimulation pulses in an open-loop fashion which do not consider the dynamics of the neural populations into account [2]. Hence, the selected DBS settings might be sub-optimal or cause

Accepted by the Research2Clinics workshop at Conference on Neural Information Processing Systems (NeurIPS 2021), .

adverse side effects [3]. Moreover, the complexities of the interactions between the nervous system and the neuromodulation systems, with large parameter spaces, pose challenges for designing and effective clinical deployment of neuromodulation technologies.

Recent advances in artificial intelligence (AI) may enable designing intelligent neuromodulation systems, that are able to learn and optimize neuromodulation control strategies autonomously, via closed-loop interaction with the nervous system. Reinforcement learning (RL) is a data-driven approach to design such intelligent closed-loop neuormodulation (iCLON) control strategies with minimal assumptions and the need for prior knowledge about the underlying physiological dynamics. These properties allow applying RL to real-world applications including iCLON control systems. However, there are many challenges in designing iCLON systems and translating them in clinical settings including algorithmic design, software implementation, hardware integration, experimental validation, and clinical deployment in implantable devices. These complexities may make designing iCLON systems out of reach for broader biomedical research community and may render designing systems that are not translatable into clinical settings.

Software implementation of iCLON algorithms requires programming expertise to translate an algorithm to semantically equivalent code for hardware implementation, while also carefully considering synchronization between an interactive prototyping environment and the algorithm. Further complicating implementation challenges are the timing, physical, and energy constraints imposed by real-time interaction with the nervous systems. Clinical implementation of the AI algorithms for iCLON systems may depend on design considerations such as power consumption, form factor, and even operational temperature which cannot be achieved by general-purpose processors and therefore require highly specialized hardware. Furthermore, prototyping and clinical testing of iCLON algorithms require translating high-level programs to operational code on the specialized hardware. Just designing a hardware module typically takes years for design experts, let alone the end users without relevant technical expertise. The hardware modules and the associated programming stack need to enable freedom in developing and adopting ever evolving and novel algorithms. This requirement is at odds with the conventional specialized hardware system design practices. These challenges make translating AI algorithms for iCLON systems currently rather infeasible.

To address these challenges, we introduce an open-source platform, dubbed Neuroweaver, for end-to-end designing, prototyping and deploying iCLON algorithms without the complexities of translating AI algorithms to implementation. Although there are various general purpose abstractions for accelerators such as OpenCL [4], CUDA [5], and Weld [6], these frameworks do not incorporate the algorithmic domain knowledge. We specifically enable the end-users to prototype iCLON algorithms in simulation environments through a Python-embedded Domain-Specific Language (DSL) framework which compiles algorithms to one or more software frameworks or target architectures, and introduce open-source, flexible accelerators capable of efficiently executing algorithm kernels. The framework is capable of taking the high-level algorithms and efficiently scheduling different compute kernels of the algorithms to different targets while also ensuring valid communication mechanisms for transferring data between frameworks.

Understanding the behavior and optimally designing the RL-based control algorithms requires interactive simulation environments where the brain in closed-loop with various candidate neuromodulation control and optimization algorithms. Due to the clinical and experimental limitations of physical interaction with the nervous system, a promising approach is to employ computational models of neural systems that enables designing, prototyping, and evaluating control algorithms before testing in in-vivo experimental setups. Leveraging mechanistic models as a surrogate of an in-vivo brain is a promising path that enables designing and prototyping various RL-based control algorithms [7, 8, 9].

The Neuroweaver platform is composed of a modular simulation environment that includes libraries of computational models of the brain stimulation, machine learning and RL/control algorithms for prototyping and experimenting with different iCLON architectures, in silico, prior to in vivo implementation. This simulation environment works in tandem with a dual software-hardware abstraction that on the front-end enables cross domain specification, while in the back-end captures the capabilities of multiple accelerators for hardware implementation.

The computational models of brain dynamics such as Bonhoeffer–van der Pol model [9] can create benchmarks for comparing the performance of state of the art RL algorithms in different neuromodulation control tasks. We specifically compare different algorithms from the perspective of challenges of integrating iCLON systems in clinical practice including sample efficiency and quality of the learned

control policies which are two important performance metrics in employing RL algorithms in clinical practice. These features are designed to emphasize the capabilities of the Neuroweaver platform for designing and prototyping RL algorithms in-silico before integrating in in-vivo experiments.

In this paper, we first introduce the simulation environment using a computation model of the brain which can be used in closed-loop with different RL algorithms. The computational model of neural population under electrical stimulation is described in section 2.1. This model is implemented in the format of OpenAI gym env [10] which has become an standard format for benchmarking different RL algorithms. We further provide a library of RL algorithms from different classes in section 2.2 for testing and prototyping their performances in a synchrony suppression task. Thereafter, to progress toward the translational challenges of integrating RL algorithms in implantable iCLON devices, a cross-domain framework enabling multi-acceleration is introduced in section 2.3.

2 Methods

2.1 Simulation environment

Pathological synchronous network activities in the brain is hypothesised as a potential source of many neurological disorders like Parkinson's disease [11]. The collective synchronous activity of neural ensembles can lead to symptoms such as tremor. DBS modulates the desired functionality of the neural systems through locally delivering stimulation to the targeted brain regions. Here, we consider to use a population of N regularly oscillating neurons, i.e. Bonhoeffer–van der Pol also known as FitzHugh–Nagumo oscillators, globally coupled via the mean field X which is implemented as an OpenAI gym environment in [9]. We utilize this environment as a simulation framework to compare and provide insights into different classes of RL algorithms. We provide a brief explanation of the model, state, action, and reward definitions in this environment and refer the readers to read [9] for more details on the environment. The regularly oscillating neurons in Bonhoeffer–van der Pol model follow the equations below:

$$\left\{ \begin{array}{l} \dot{x} = x_k - \frac{x_k^3}{3} - y_k + I_k + \epsilon X + A \\ \dot{y} = 0.1(x_k - 0.8y_k + 0.7), \end{array} \right\} \tag{1}$$

where $X = \frac{1}{N}\sum_{k=1}^{N} x_k$ is the mean field, A is the action stimuli applied to each individual neuron $k = 1, \ldots, N$ of the total N neurons. Actions, are considered to be ideal δ-shaped pulses with the amplitude $-A_{max} \leq A_t \leq A_{max}$ which gets updated at each time step $t_n = n\Delta$ and Δ is the sampling rate of the environment.

The state of the environment at time step t is the value of the mean field model, i.e. $X(t)$. We used the exponential reward function as in:

$$R(t) = exp^{(-(X(t)-<X_{state}>)^2-\beta|A_t|)}. \tag{2}$$

In equation 2, $< X_{state} >_t = \frac{1}{M}\sum_{l=1}^{M} X(t-l+1)$ consists of M most recent values of the mean filed and it is considered to account for the oscillatory activity of the neural populations. The total energy supplied to an ensemble of neurons is a measure that we aim to minimize in practical DBS settings and the second term in equation 2 is added in favor of minimizing the total stimulation energy.

2.2 RL algorithm library

We provide a library of multiple classes of RL algorithms including model-free, model-based, on-policy, and off-policy RL algorithms for testing in closed-loop with the computational model described in section 2.1. We show the utility and the extensibility of this in-silico simulation environment in providing insight on the behavior of RL algorithms in the context of a neuromodulation tasks in terms of speed of convergence and the quality of learning the optimal control policies which are two important performance metrics in employing RL algorithms in clinical practice.

A standard RL task can be formulated as a Markov Decision Process (MDP) [12] defined by a tuple (S, A, r, T, P), where S and A are state and action spaces, r is a reward function, T is the set of terminal conditions, and P is the state transition probability. The general goal of reinforcement

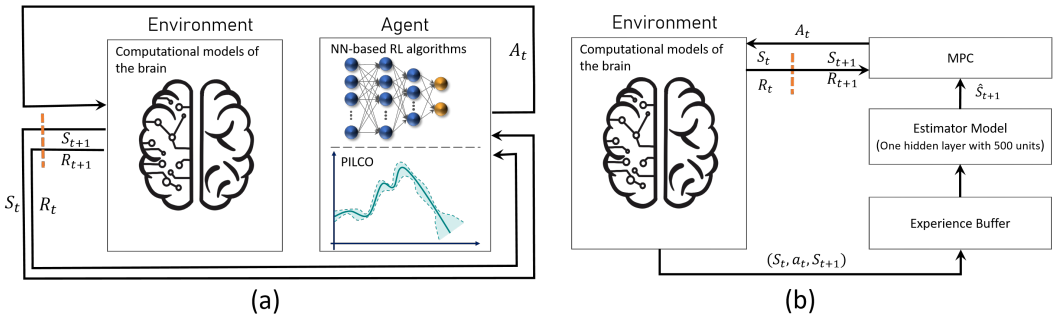

Figure 1: The modular architecture of the Neuroweaver simulation environment including the computational model of the neural population under electrical stimulation and the RL agents; (a) neural network-based model-free algorithms and PILCO, (b) Model-based RL with MPC.

learning algorithms is to find the optimal policy π maximizing the discounted cumulative expected reward. High-level explanations of the algorithms is provided in the following sections and Fig. 1 shows high-level overview of the modular simulation environment including the computational model in closed-loop with RL agents.

2.2.1 Model-free reinforcement learning algorithms

We deploy three model-free RL algorithms in closed loop with the simulation environment, including proximal policy optimization (PPO) [13], soft actor-critic (SAC) [14], and Deep Deterministic Policy Gradient (DDPG) [15] to evaluate the feasibility of utilizing model-free RL approaches in intelligent closed-loop neuromodulation control systems. The advantage of model-free RL algorithms over more complex methods is that they do not rely on constructing a sufficiently accurate environment model and hence, their performance are not affected by model bias.

These three algorithms can be divided into two main categories: on-policy (e.g., PPO) and off-policy (e.g., SAC and DDPG). PPO is a policy gradient method that has shown high quality of performance in many applications. Optimization of policy parameters are handled using the gradient descent algorithm. We employed a variant of PPO, called PPO-clip, that relies on clipping the objective function to keep the new policy close to the old policy. PPO is applied in an actor-critic framework. The actors maps the state to an action and the critic gives an expectation of the agent's reward with its corresponding state. The policy is updated via a stochastic gradient ascent optimizer to ensure the exploration while the agent will gradually tend to exploit what it has learned over the course of training.

SAC and DDPG are off-policy approaches that has been shown to be generally more sample efficient than on-policy methods since the off-policy agents utilize the replay buffer containing the old experiences in contrast to the on-policy agents. As the name suggests SAC is also an actor-critic algorithm. Instead of only considering the expected reward, SAC is trained to maximize a trade-off between expected reward and entropy. DDPG is a deep variant of the deterministic policy gradient algorithm, which can also be viewed as an actor-critic algorithm, using a Q-function estimator to enable off-policy learning, and maximizing this Q-function by an actor. Since DDPG is a deterministic policy, we add adaptive noises to the parameters of the neural network to encourage exploration [16]. Since on-policy methods attempts to improve the most recent policy, they tends to be generally more stable compared to the off-policy methods.

2.2.2 Model-based reinforcement learning with model predictive control (MPC)

To improve sample efficiency which is a critical factor in iCLON systems, we investigate model-based RL algorithms. The first method is a combination of a model-based RL algorithm with model predictive control (MPC). Fig. 1(b) shows the high-level overview of this method that consists of two components: learning the underlying dynamics of the environment, and using a MPC controller to plan and execute actions. To approximate the state transition model of the simulation environment, we initially collect random trajectories and add the history of collected samples to the experience

buffer. The estimated dynamical model $\hat{f}$ is formulated as $\hat{s}_{t+1} = s_t + \hat{f}_\theta(s_t, a_t)$, where s_t and a_t are the state and action at step t respectively, following the setting in this work [17]. We used as a neural network to model the dynamics, where the parameter vector θ represents the weights of the neural network, aiming to minimize the mean squared error $\xi = 1/D \sum_{(s_t,a_t,s_{t+1})\in D} ||\delta - \hat{\delta}||^2$ between the observed difference of two consecutive time steps, i.e. $\delta = s_{t+1} - s_t$, and the model predictions, i.e. $\hat{\delta} = \hat{f}_\theta(s_t, a_t)$. After initialization, MPC selects the next actions to be evaluated with the goal is to minimize the cost function to achieve the synchrony suppression task. The cost function is in the same format as the reward function in equation 2 with the negative sign and the different value of β.

2.2.3 Probabilistic Inference for Learning Control (PILCO)

Probabilistic inference for learning control (PILCO) [18] is model-based data-efficient approach to policy search without considering any prior domain knowledge about the underlying dynamic. Model-based RL approaches often assume that the learned dynamics model is sufficiently accurate which will lead to low performance in the presence of model bias. Model bias is particularly an issue in cases where there is limited prior knowledge or limited data available. PILCO employ Gaussian process (GP), a non-parametric probabilistic model [19], that takes the model uncertainty into account to address the model bias issue. The main advantage of PILCO is that it remarkably improves the sample efficiency in continuous state-action spaces which sets the pathway of integration of PILCO in closed-loop clinical settings and experimental setups.

Consider the following dynamical system $x_t = f(x_{t-1)}, u_{t-1})$, Where f is the unknown state transition function with continuous state, x, and action, u, domains. The goal of PILCO is to find a deterministic policy that maximizes the expected return or minimizes the expected cost, $c(x_t)$ of following the policy π over the time horizon T as in $J_\pi(\theta) = \sum_{t=0}^{T} E_{x_t}[c(x_t)], x_0 \sim N(\mu_0, \Sigma_0)$. PILCO assumes that π is a function parametrized by θ and that the cost function $c(x)$ encodes some information about a target state x_{target}. We used the squared exponential cost function.

PILCO employs GPs as a probabilistic approach to model the underlying dynamics. The model uncertainty is used for planning and policy evaluation steps. PILCO evaluates the policy using the deterministic approximate inference method which enables policy improvement through analytic policy gradients. Analytic policy gradient is more efficient than estimating policy gradients with sampling and enables using standard non-convex optimization methods like LBFGS to find the optimal policy parameters. Here, the learned state-feedback controller is the nonlinear radial basis function network as described in [18], where the parameters of the RBF network controller is optimized using LBFGS optimization. We empirically show that PILCO learns the synchrony suppression task with consuming considerably less samples comparing to other methods. This makes PILCO a suitable candidate for being integrated in iCLON systems.

2.3 Neuroweaver Platform

Designing, prototyping, and experimenting with neuromodulation control systems requires implementing closed-loop analytic pipelines using interoperable modules. These systems can be modeled as several interacting services in computational environments that are often not limited to a single domain as shown in Fig. 2. Closed-loop neuromodulation pipelines include several analytic steps that employ signal processing, Machine learning (ML)/deep learning (DL), and RL approaches. However, most of these approaches are computationally expensive, and rely on compute kernels spanning multiple domains of algorithm as well as multiple compute stacks, making integration into iCLON systems an arduous task and implementation of these pipelines in resource-constrained computational infrastructures such as implantable devices infeasible. To bridge the transnational gap and enable hardware implementation of iCLON systems a framework capable of acceleration of a cross-domain application on different accelerators, called cross-domain multi-acceleration, is required.

To enable programmers to readily develop cross-domain applications using multiple accelerators on a FPGA, we devised Neuroweaver, a full-stack framework comprised of a front-end which is a Domain-Specific Language (DSL) for cross-domain algorithmic specification and a back-end which describes the capabilities of domain-specific accelerators. By delineating between front-end algorithm and the possible back-end targets for the hardware implementation of that algorithm, cross-domain

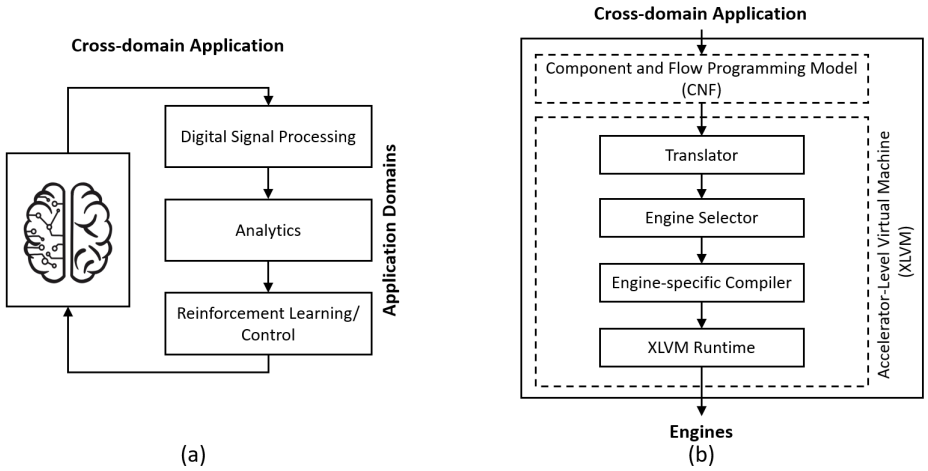

Figure 2: Overview of Neuroweaver platform; (a) A cross-domain closed-loop neuromodulation pipeline. (b) Neuroweaver workflow to enable cross-domain multi-acceleration.

end-to-end closed-loop neuromodulation applications can be compiled to multiple heterogenous accelerators.

2.3.1 Abstract Domains and Engine Specifications

Neuroweaver distinguishes between front-end algorithm specification and back-end execution targets by using "Abstract Domains" which are libraries of target-agnostic algorithms associated with a specific domain, and "Abstract Engines" which are implementations of algorithms for different accelerator targets. The combination of these two constructs delineates between the semantics of an end-to-end application (front-end) and the possible ways the application can be executed (back-end).

To use an "Abstract Domain", domain experts independently pre-define common capabilities and used in applications for an algorithm domain. The collection of pre-defined capabilities are called "Domain Descriptions", which can be imported and used to implement applications. As an example, Faster-Fourier Transform (FFT) is an algorithm commonly used in the domain of digital signal processing (DSP), and would therefore be included as a capability in the DSP domain description for use in application implementation.

The counterpart to these domain descriptions are "Abstract Engine's", which are sets of capability implementations for a specific compute target. These abstract engine's can be defined for either software or hardware. For instance, an engine can be defined for a software library running on CPU or for code targeting an accelerator. Together, these constructs are combined to enable users to define and compile end-to-end applications using a DSL as it's programming interface.

2.3.2 Programming Interface

Neuroweaver's programming interface is designed to enable utilizing multiple heterogeneous platforms in a seamless manner to run the application program in an end-to-end fashion. This enables utilization of hardware accelerators for compute-intensive kernels in the application.

Neuroweaver applications are defined using a Component and Flow Annotation Model (CNF), where Components and Flows represent the computation and dataflow between components, respectively. In particular, a "Component" is a language construct that is explicitly used within the code, whereas the "Flow" is implicitly present in between components. The CNF enables programmers to use a set of light-weight language annotations to delineate various components of their end-to-end applications to be targeted for acceleration.

Note that each component defined in the CNF program represents one module of the closed-loop neuromodulation control system interacting with one another. CNF program enables designing closed-loop neuromodulation control systems in form of interoperable and translatable modules. One simple example of the lightweight language annotation used in the CNF DSL is shown in Fig. 3 and

the end-users of Neuroweaver needs to deal with. This example shows the definition of Components

```
1 # Component definition
2 with Component(outputs=[c]) as comp_A:
3   a = 1 + 2
4   b = a + 2
5   c = b - 2
6
7 # Component definition
8 with Component(inputs=[a]) as comp_B:
9   b = a + 1
10  c = b - 10
11  print(c)
12
13 # Component instantiation
14 comp_A_out = comp_A()
15 comp_B(comp_A_out)
```

Figure 3: A simple exaple of the programming inteface with lightweight language annotations. This examples shows both the definition of Components and instantiation of Components.

and instantiation of Components. Lines 2-11 include two examples of Component definitions, and Lines 14-15 are examples of instantiation of these Components. In this example, there are two Components, “comp_A" and “comp_B", that are communicating with each other. Every Component definition has a body which corresponds to the computation that needs to be done. Each Component is marked by the “with" block in Python. Line 2 is an example of how to delineate a Component. It states that this Component is named “comp_A", and it has one output called “c" which appears in Line 5 of the Component body. Lines 3-5 correspond to the body of the “comp_A" Component. The “comp_B" Component is another example of Component definition, where it takes one input, called “a", which is used in the Component body in Line 9.

Lines 14-15 shows instantiation of these two Components. When instantiating Components, the Component name used in the definition must be used, as done in Line 14 for the “comp_A" Component. If the Component definition includes an output, it can be expressed as a variable assignment, as done in Line 14 (“comp_A_out"), and this variable name does not need to be identical to the output variable name used in the Component definition (“c" in Line 2). Line 15 shows an instantiation of a Component which is defined to take one input, which in this case is “comp_A_out" - the output of “comp_A" Component.

2.3.3 Neuroweaver Workflow

Once the CNF program is written and parsed by the Neuroweaver compiler, the execution workflow involves the following steps: (1) Select capability implementations from different Engine Specifications; (2) Invoke the target engine-specific compiler; and (3) Execute the Accelerator-Level Virtual Machine (XLVM) runtime system. A general overview of Neuroweaver workflow is provided in Fig. 2(b).

Engine selection consists of mapping the capabilities used in CNF programs to various accelerators by selecting the corresponding implementation from one of the different Engine Specifications. As mentioned, users define CNF programs by importing domain descriptions and using their capabilities. Initially, a CNF program and the capabilities it uses are target-agnostic, but are capable of targeting one of the different engines which provide an implementation. This allows for optimization of target selection by using the different engine properties and communication requirements to explore different performance-cost tradeoffs.

Once the optimal engine targets have been selected, the compiler for the different engines used in the CNF program can be used to generate executable code for the target platform. The compiler for each engine generates a binary file that is executed on the host CPU at runtime and orchestrates the initiation, execution, and termination of accelerators. The canonical set of operations in engine executables constitutes loading the input data to engines, setting the configuration registers, triggering the computation, observing the runtime status, and receiving the output data.

Once compilation steps are complete, the next step is to let Neuroweaver’s runtime system handle end-to-end execution of the application program while seamlessly handling the offloading of Com-

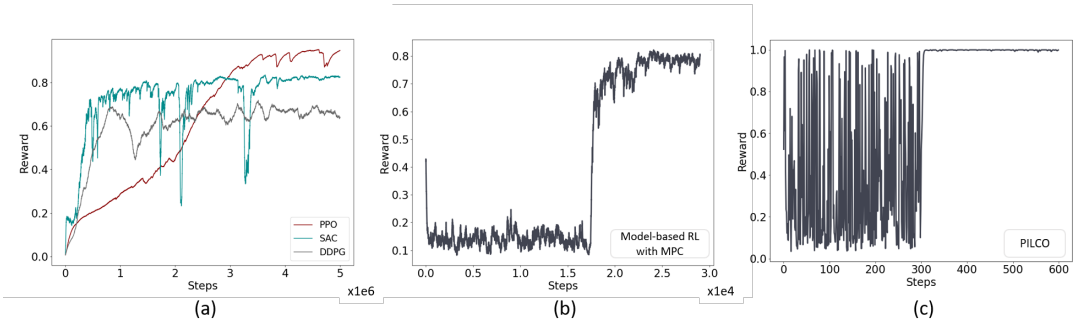

Figure 4: Learning performance of different RL algorithms. (a) Reward function of

ponents to their respective target acceleration platforms. We call the runtime system XLVM, short for Acceleration-Level Virtual Machine. Both the compiler and runtime system aim to schedule Components to run on an acceleration platform (through a process called Engine Selection), with the objective of improving the overall performance of the end-to-end application. However, if it turns out that the Component does not have an acceleration platform to run on, it will default to running on the CPU, which we assume to be available throughout this project. This provides an extra level of flexibility in case a runtime failure occurs. The final goal is to make compilation and execution steps automated to provide a better experience for the end users from the neuroscience community.

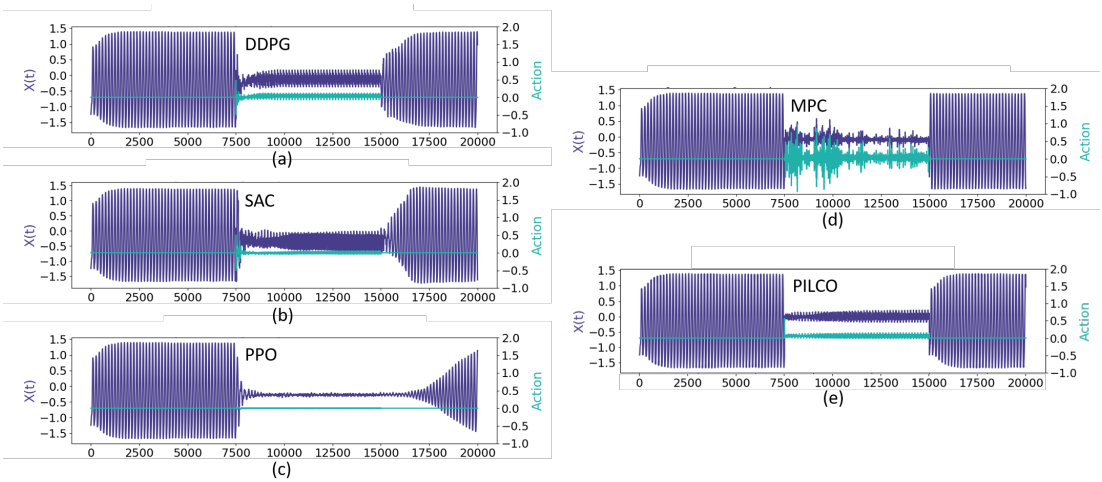

Figure 5: Final performance of different RL algorithms in the synchrony suppression task.

3 Results

We have created a suite of simulation environments using a biophysical models of brain stimulation to design and test intelligent closed-loop neuromodulation systems. We employed the mechanistic model of the population of N regularly oscillating neurons, i.e. Bonhoeffer–van der Pol model implemented in the format of OpenAI gym which is a standard environment for designing intelligent agents. Using this setup, we demonstrated in-silico experiments to design iCLON systems using the state-of-the-art RL algorithms for suppressing pathological synchrony. We evaluated the performance of five different RL algorithms in the synchrony suppression task in terms of the quality of learning the task and sample efficiency which are two key factors in designing iCLON algorithms. We used the stable baseline library for implementation of the model-free RL algorithms, i.e. PPO, SAC, and DDPG. In our implementations, we used two hidden layers MLPs with 64 neurons for all three model-free RL algorithms. Fig. 4(a) shows the reward function during the training phase of the three model-free algorithms. As shown in Fig. 4(a), SAC and DDPG which are off-policy algorithms converge faster compared to PPO which is an on-policy method. However, PPO achieves a higher

reward and a better final performance in learning the synchrony suppression task at the expense of more interaction with the neural environment. The quality of performing the task after convergence is shown in Fig. 5(a)-(c), where the first 7500 samples are showing the oscillatory behavior of the neural populations without taking any action, i.e. applying any stimulation pulses. The middle 7500 samples is showing the neural states by taking actions with the learned RL policies, and the last 7500 samples show that the population of neurons start synchronizing again if we stop the intervention.

We further expanded the library of algorithms by adding two model-based approaches to improve sample-efficiency. The first approach is model-based RL with MPC. In our implementations we used a single-layer MLP with 500 neurons for modeling the underlying neural dynamics. Fig. 4(b) show the reward value of the MPC approach. Although model-free methods require at least $1e6$ samples to converge, the model-based RL with MPC has shown improvement in terms of sample efficiency and converges at around $2.5e4$ steps. However, there is still room for improvement in terms of its final performance in the synchrony suppression task as depicted in Fig. 5(d) and having $2.5e4$ interactions with the nervous system might still be impractical for in-vivo experiments.

The next model-based RL algorithm that we tested is PILCO. As shown in Fig. 4(c), after a random initialization phase of length 300 steps, the RL agent quickly converges and learns the synchrony suppression task as shown in Fig. 5(e). Although the best final performance in terms of synchrony suppression and minimizing the power of actions is achieved by PPO, but that is at the cost of having at least $3e6$ interactions with the environment that is not practical for being integrated in iCLON systems in clinical practice. On the other hand, PILCO shows a noticeable improvement in sample efficiency at a cost of consuming slightly higher action power and slightly higher level of synchrony which is much more well-suited for clinical practice. In general, our evaluations support the hypothesis that RL algorithms are capable of handling the decision-making process in closed-loop neuromodulation control systems. In addition, we showed the utility of simulation environment in designing and prototyping iCLON systems in silico before testing in in-vivo experiments.

We further implemented the closed-loop simulation with PILCO using the component and flow annotation model, compiled, and executed the closed-loop pipeline using Neuroweaver workflow to compute the compilation and run time overheads introduced by the Neuroweaver platform. We used a pure python implementation of the closed-loop simulation with PILCO as our baseline. Based on our simulations, the baseline execution time for 500 is iterations is 118.68 seconds, while the total run time with Neuroweaver platform is 119.37 seconds. The total compile and run time fir Neuroweaver is 119.47 seconds. The results show that the compilation overhead which is a one time process takes 0.1 seconds, and the execution for 500 iterations introduces 0.69 seconds overhead which is negligible. Enabling hardware acceleration with minimal complexity of translating the algorithms to implementation for the end-users is a work in progress. We are working on enabling hardware acceleration using the libraries of target-agnostic algorithms associated with a specific domain and move towards bridging the gap for implementing AI-based iCLON algorithms in implantable devices.

4 Discussion

We presented an open-source platform, dubbed Neuroweaver, for end-to-end designing, prototyping and deploying translatable intelligent closed-loop neuromodulation systems. The main purpose of Neuroweaver is to bridge the translational gap between designing and deploying clinically useful iCLON systems from multiple perspectives. First, it provides a simulation environment using computational model(s) of the neural systems for designing, testing, and prototyping closed-loop neuromodulation systems before deploying in clinical or in in-vivo experimental settings. Second, it provides libraries of algorithms from multiple domains including RL, signal processing and machine learning to enable modular design of closed-loop pipelines. Finally, Neuroweaver enables cross-domain multi-acceleration to progress towards developing implantable iCLON systems.

We demonstrated the utility of Neuroweaver in experimental design using a computational model of the brain under electrical stimulation. We evaluated the Performance of different classes of RL algorithms, including model-based, model-free, policy gradient and Q-learning, on-policy, and off-policy, in a synchrony suppression task. We assessed the utility of employing the RL algorithms in iCLON systems considering two metrics including speed of convergence as in Fig. 4 and the quality of learning task as shown in Fig. 5. This model represented an example of physiological models that can be used for designing iCLON systems. The Neuroweaver platform is capable of incorporating

different computational models as surrogates of the target physiological system. These models may include mechanistic biophysical models or data-driven models that are built from the experimental data.

Although we showed the utility of Neuroweaver in experimental design using the simulation framework, we still need to expand the simulation environment with both computational and data-driven models of neural systems to enclose a wider range of neuro-physiological experiments and define a wider range of tasks and control objectives other than synchrony suppression. We compared different algorithms from the perspective of sample-efficiency and the quality of the learned task (maximum achievable reward). Although the mentioned performance metrics are critical for designing iCLON systems, there are more real-world performance metrics that needs to be considered for better designing iCLON systems including safety and robustness to noise amongst other metrics.

We implemented a sample closed-loop simulation using the component and flow annotation model, compiled, and executed the simulation using Neuroweaver workflow. Our results showed that Neuroweaver workflow does not add a significant computing overhead to the execution of the closed-loop pipelines. This is an important factor especially for the real-time implementation of iCLON algorithms. Hardware integration using the Neuroweaver platform is still a work in progress.

5 Conclusion

In this study, we presented our work toward an open-source AI-based platform, called Neuroweaver, for designing intelligent closed-loop neural systems to bridge the translational gap between research studies and clinical practice. Neuroweaver not only provides a simulation environment for modular designing and prototyping closed-loop neuromodulation pipelines, but also minimizes the complexities of translating the algorithm to implementation through a Python-embedded DSL framework which compiles algorithms to one or more software frameworks or target architectures. We showed the utility of Neuroweaver in designing and prototyping iCLON systems using RL algorithms. Moreover, our results showed that implementation and execution of the closed-loop simulations using Neuroweaver workflow does not add execution overhead which is critical for real-time implementation of iCLON interfaces. Integrating hardware acceleration with minimal complexity of translating the algorithms to implementation for the end-users is a work in progress to make the platform usable for a wide range of clinical applications.

References

[1] Philip A Starr, Jerrold L Vitek, and Roy AE Bakay. Deep brain stimulation for movement disorders. *Neurosurgery Clinics of North America*, 9(2):381–402, 1998.

[2] Anders Christian Meidahl, Gerd Tinkhauser, Damian Marc Herz, Hayriye Cagnan, Jean Debarros, and Peter Brown. Adaptive deep brain stimulation for movement disorders: the long road to clinical therapy. *Movement disorders*, 32(6):810–819, 2017.

[3] Sierra Farris and Monique Giroux. Retrospective review of factors leading to dissatisfaction with subthalamic nucleus deep brain stimulation during long-term management. *Surgical neurology international*, 4, 2013.

[4] Aaftab Munshi, Benedict Gaster, Timothy G Mattson, and Dan Ginsburg. *OpenCL programming guide*. Pearson Education, 2011.

[5] CUDA Nvidia. Compute unified device architecture programming guide. 2007.

[6] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Matei Zaharia, and Stanford InfoLab. Weld: A common runtime for high performance data analytics. In *Conference on Innovative Data Systems Research (CIDR)*, volume 19, 2017.

[7] Meili Lu, Xile Wei, Yanqiu Che, Jiang Wang, and Kenneth A Loparo. Application of reinforcement learning to deep brain stimulation in a computational model of parkinson's disease. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(1):339–349, 2019.

[8] BA Mitchell and LR Petzold. Control of neural systems at multiple scales using model-free, deep reinforcement learning. *Scientific reports*, 8(1):1–12, 2018.

[9] Dmitrii Krylov, Remi Tachet, Romain Laroche, Michael Rosenblum, and Dmitry V Dylov. Reinforcement learning framework for deep brain stimulation study. *arXiv preprint arXiv:2002.10948*, 2020.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[11] Constance Hammond, Hagai Bergman, and Peter Brown. Pathological synchronization in parkinson's disease: networks, models and treatments. *Trends in neurosciences*, 30(7):357–364, 2007.

[12] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on machine learning (ICML-18)*, pages 1861–1870. Citeseer, 2018.

[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Yuval Tassa Tom Erez, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[16] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[17] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[18] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[19] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.