PATIENT-OPTIMIZED TEMPORALLY-ADAPTIVE NEUROSTIMULATION FOR EPILEPSY USING DEEP-EDMD KOOPMAN MPC

ROJIN SALAHI

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2024

ABSTRACT

This thesis presents the design and implementation of a closed-loop neurostimulator controller for delivering patient-optimized, temporally-adaptive stimulation pulses to control epileptic seizures. The method employs a predictive model and a Model Predictive Controller (MPC) to optimize stimulation. The predictive model, based on a deep learning extended dynamic mode decomposition algorithm, approximates the Koopman operator, capturing a patient's brain dynamics and forecasting stimulation efficacy. The MPC uses these predictions to converge rapidly to an optimal set of stimulation parameters. The system is capable of adapting to changes in brain dynamics over time, ensuring continuous optimization. Both the predictive model and MPC are implemented in software and on hardware (FPGA and ASIC synthesis), achieving a power consumption of 97.09 μW.

Additionally, we present a framework incorporating a neural mass model (NMM) fine-tuned to patient pre-recorded data. The NMM generates synthetic intracranial electroencephalography (iEEG) highly correlated with real iEEG during normal and seizure periods. This framework is used to test and validate other patient-optimized, temporally-adaptive stimulation approaches. Our temporally-adaptive stimulation optimization for seizure control was validated using this framework.

ACKNOWLEDGMNET

First and foremost, I would like to sincerely thank my supervisor, Dr. Hossein Kassiri. Working with Dr. Kassiri over the past two years has been a truly rewarding experience. His expertise and guidance have been instrumental in shaping my research, and I have gained invaluable insights under his mentorship. His continued support and trust in my capabilities mean so much to me, and I am deeply grateful for his patience and encouragement throughout this journey.

I would like to express my gratitude to my committee members, Dr. Amir Sodagar and Dr. Kohitij Kar for their great feedback and valuable comments on this thesis.

I would also like to thank my colleagues and friends, Sayedeh Mina Sayedi, Nashmil Mansouri, Abdul Muneeb, Aref Molaei and all others who have supported me along the way, for making my master's studies a memorable experience.

Finally, I want to express my deepest gratitude to my beloved parents for their unwavering encouragement and support throughout my life. Your belief in me has been a constant source of strength, and I am forever thankful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

1.1 Epilepsy and Its Treatment

Epilepsy, one of the most debilitating neurological disorders, affects approximately 1% of the global population. It is the second leading cause of mental health disability, particularly among young adults [1], and contributes to a global disease burden comparable to that of breast cancer in women and lung cancer in men [2]. It is characterized by the brain's recurrent transition into a seizure—a brief state marked by excessive neural activity, varying degrees of synchronization and desynchronization, and oscillatory dynamics [3-4]. Seizures can cause changes in behavior, movements, feelings, and levels of consciousness. These uncontrolled burst of electrical activity can vary in duration and frequency, ranging from a few seconds to minutes and occurring anywhere from a few times a month to several times a day.

Epilepsy and seizures vary significantly among patients, with differences in onset, seizure type, and underlying causes. They are generally classified into two main types: focal and generalized [5-6]. Focal seizures originate in a specific area of the brain and can affect

awareness or manifest through specific motor or non-motor symptoms. Generalized seizures, on the other hand, involve the entire brain and can include motor symptoms like convulsions or non-motor symptoms such as absence seizures. The causes of epilepsy are diverse and can be categorized into groups such as genetic, structural, metabolic, infectious, immune-related, and unknown origins [7].

These variations in seizure types and causes mean that treatment responses can differ greatly between patients. This suggest that managing epilepsy effectively requires a personalized approach that considers the specific characteristics of each patient's condition. By tailoring treatments to the individual's unique needs, healthcare providers can better suppress seizures and improve the patient's overall quality of life.

The primary treatment approach for individuals with epilepsy is medication, but when it proves ineffective, these patients are considered medically refractory. Approximately half of the medically refractory patients are not found to be candidates for a resective surgery. For the other half, the only viable treatment option is electrical neurostimulation, which has been widely investigated and has shown promising effectiveness over the past two decades [8-12].

Brain neurostimulation involves the use of electrical pulses to either excite or inhibit neural activity near the stimulation electrodes, aiming to modulate seizure-like neural oscillations to a seizure-free state. It can be applied to the brain using various modalities (e.g., magnetic, electrical) with different levels of invasiveness (i.e., non-invasive, minimally

invasive, fully invasive) and at different application points (e.g., peripheral nervous system such as the vagus nerve or directly to the brain).

1.2 Non-Invasive and Invasive Stimulation

Popular non-invasive approaches include trans-cranial magnetic stimulation (TMS) and trans-cranial direct current stimulation (tDCS). TMS modulates brain activity by producing robust magnetic fields capable of inducing electrical currents within neural tissue. With strengths of 1-3 T at the coil surface, TMS offers versatile applications including single-shot stimulation, repetitive sessions (rTMS), or targeting deeper brain regions (dTMS). rTMS uses high-frequency pulses at 5-20 Hz for excitatory effects or low-frequency pulses at 1 Hz for inhibitory effects [13-14]. tDCS delivers a low-intensity, continuous electrical current (1-2 mA) via scalp electrodes to target specific brain regions, for durations varying from several minutes to up to an hour [15]. Despite significant progress in both methods, they are not yet approved for epilepsy treatment, and are used experimentally in the US and EU, necessitating further research to establish their efficacy in managing drug resistant epilepsy.

Among invasive approaches, vagus nerve stimulation (VNS) and deep brain stimulation of the anterior nucleus of the thalamus (DBS-ANT) are two approved neuromodulation techniques for managing epilepsy. VNS has received FDA approval for focal epilepsy in both adults and children aged 4 years and older, with the latest approval in 2017 for focal epilepsy

and 2018 for generalized epilepsy. On the other hand, DBS-ANT is FDA approved for focal epilepsy in adults aged 18 and older, with approval granted in 2018 [16-19]. VNS devices intermittently stimulate the left vagus nerve, affecting brainstem projections and leading to changes in blood flow and neurotransmitters, particularly in the thalamus. DBS-ANT involves intermittent stimulation using depth electrodes to target the bilateral anterior nucleus of the thalamus. The programming parameters for VNS and DBS-ANT, including current, frequency range, and duty cycle, can vary based on individual patient needs and responses.

1.3 Open-Loop vs Closed-Loop Stimulation

Beside modalities and invasiveness, an important way to categorize neuro-stimulators is based on their strategy being open-loop or closed-loop.

1.3.1 Open-Loop Stimulation

Current commercial neurostimulation devices typically use open-loop actuation strategies, where stimulation is delivered without considering the current brain state. In this approach, periodic pulses with pre-programmed parameters are injected into the brain, as shown in Figure 1-1. However, because this method does not adapt to the brain's activity, it faces significant challenges. For example, Applying stimulation too frequently can lead to tissue damage and increased energy consumption, while applying it too infrequently may miss

the onset of seizures. The lack of responsiveness to the brain's changing conditions makes it difficult to strike the right balance, resulting in either overstimulation or inadequate seizure control.

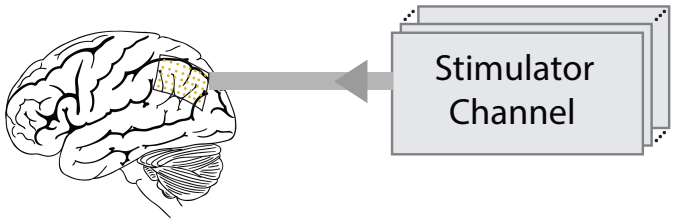

Figure 1-1 Open-Loop Neurostimulation System

One of the commercial available open-loop neurostimulation systems is the Medtronic Activa RC which is a Deep Brain Stimulation (DBS) system (Figure 1-2(a)) that offers a long battery (rechargeable) life of 9-15 years, with the requirement for regular charging sessions every 1-2 weeks. This device received FDA approval for epilepsy treatment in 2018 for patients with focal epilepsy aged 18 years and older. Another prominent player in the market is the Boston Scientific Vercise system (Figure 1-2(b)). Available in both primary cell and rechargeable versions, the Vercise system features segmented leads, current shaping, and multiple independent current control. It has a battery life of 4-5 years and offers precise stimulation comparable to the Medtronic system.

The LivaNova AspireHC (Figure 1-2(c)) is a Vagus Nerve Stimulation (VNS) device known for its effectiveness in managing drug-resistant epilepsy. FDA-approved initially in 1997 for both focal and generalized epilepsy, and for patients as young as 4 years old, it provides reliable long-term therapy with a median battery life of 4-6 years. Finally, a recent example in

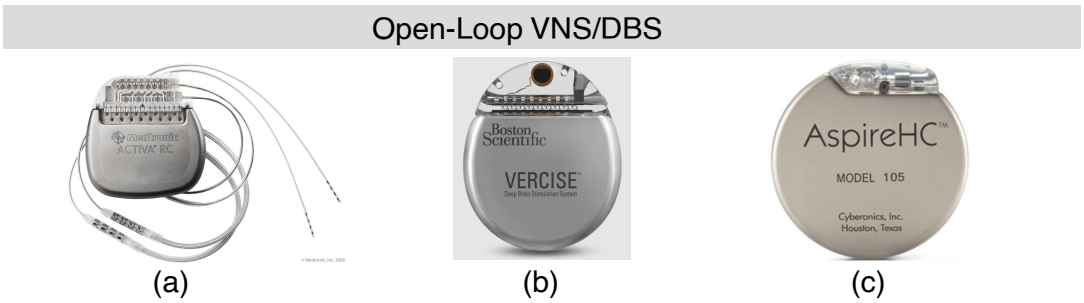

Figure 1-2 Examples of commercially available open-loop devices.

this category is the Precisis EASEE II, which is a subcutaneous neurostimulator that employs 5 subscalp electrodes placed above the seizure focus for high-frequency or direct current stimulation. This system offers an innovative approach to neuromodulation therapy, targeting the seizure focus directly.

1.3.2 Closed-Loop Stimulation

Motivated by the problems associated with open-loop systems, closed-loop neurostimulation is increasingly gaining attention in research and clinical applications. As shown in Figure 1-3, the closed-loop neurostimulation system consists of a recording block, signal processing, and a stimulator. This approach enables us to gain insight into brain activity by recording signals from the brain, detecting or predicting the occurrence of neurological events (e.g., seizures) in the signal processing block, and delivering stimulation only when necessary, i.e., upon event detection. The on-demand responsive operation based on real-time monitoring results in a significantly lower number of stimulations compared to the open-loop strategy, hence leading to enhanced safety, less severe side effects, and better energy efficiency.

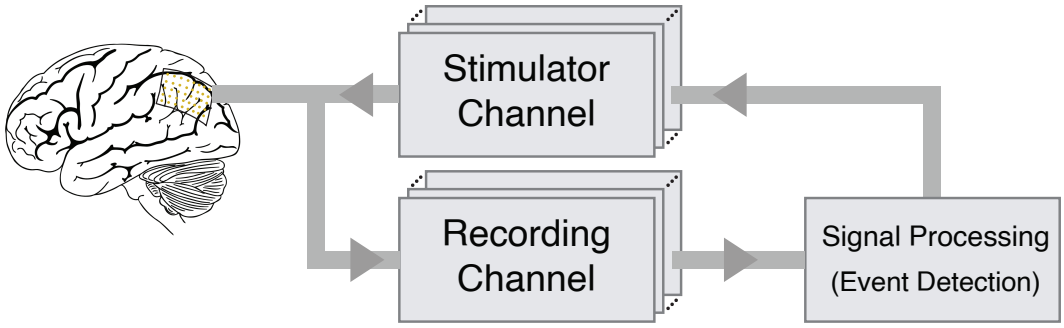

Figure 1-3 Closed-Loop neurostimulation system.

More importantly, it results in a better clinical efficacy, mainly attributed to precise timing in seizure suppression. As shown in Figure 1-4(a), stimulating before the seizure is fully developed substantially enhances the likelihood of seizure suppression, as early intervention is believed to halt the progression of the seizure [20]. Conversely, Figure 1-4(b) illustrates that despite multiple stimulations, seizures could not be suppressed after it was fully evolved. Further evidence with statistical significance is shown in Figure 1-4(c) and (d) (from [21] and [22]), which compare the efficacy of open loop and responsive closed-loop stimulation on a number of subjects, highlighting the superior performance of the closed-loop method. Mounting evidence, including the presented data from rodent models of epilepsy, as well as data from experiments on humans [23], reveal substantial reductions in daily seizure occurrences with closed-loop approaches, reaffirming their superiority in reducing seizure frequency, mainly due to the importance of timing.

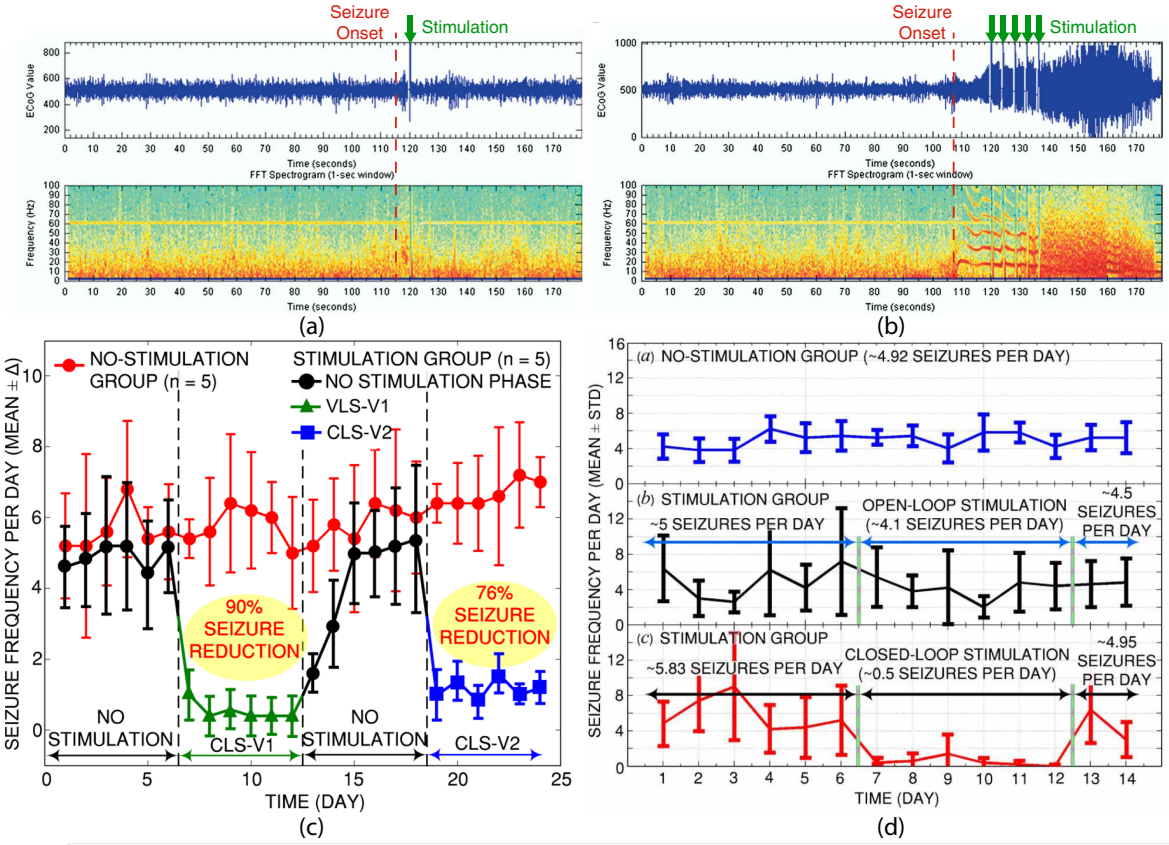

Figure 1-4 Effectiveness of early and closed-loop stimulation in reducing seizure frequency compared to late and open loop stimulation [20-23].

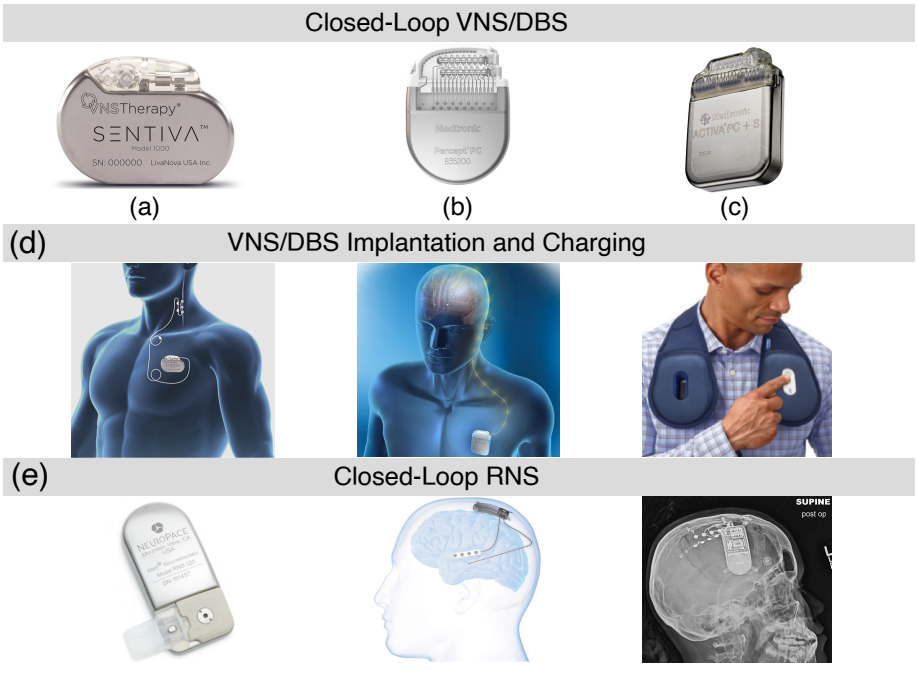

Figure 1-5 Example of market closed-loop.

Several closed-loop devices are available in the market, offering innovative approaches to seizure management. One notable example is the closed-loop VNS device "SenTiva" from LivaNova (Figure 1-5(a)), which employs an adaptive stimulation strategy – called AutoStim mode – in which they dynamically adjust stimulation parameters, such as magnitude, frequency, and duty cycle, based on physiological indicators like tachycardia (i.e., heart rate exceeding 100 beats per minute). By modulating stimulation in response to physiological cues, these devices aim to enhance effectiveness, albeit through a different mechanism than traditional closed-loop systems [24-25]. Sentiva M1000 received FDA approval in 2017 and is capable of scheduling programming.

Another noteworthy closed-loop neurostimulation device is the cortical stimulator from Medtronic called Percept PC (Figure 1-5(b)). This device records neural signals from two channels and incorporates spectral analysis capabilities. It is capable of recording local field potentials (LFP) at seizure focus areas such as the ANT region [26]. It stores 10-minute LFP power data for a user-selected 5Hz frequency band, a feature known as chronic sensing. However, despite its potential for closed-loop functionality, the Percept PC does not currently support responsive stimulation triggered by seizure detection. Instead, it operates based on pre-programmed stimulation settings. Other variant of this device from Medtronic is the Activa PC+S and Summit RC+S system (Figure 1-5(c)), which employs spectral analysis and a linear classifier, with recording and wireless capabilities.

The NeuroPace Responsive Neurostimulator (Figure 1-5(d)) stands as a pioneering example of closed-loop stimulation, widely regarded as the most prominent clinical device in this category. It employs an intracranial-EEG-based approach to seizure detection and stimulation. Implanted within the scalp, this device features both surface and depth electrodes for neural signal acquisition, and is capable of recording electrocortical data from four channels, allowing for monitoring brain activity with spatial information.

The device is capable of on-chip processing and real-time analysis of neural patterns, employing features such as line length, low-magnitude high-frequency activity detection, and half-wave detection at specific frequencies for accurate preictal detection. Upon detecting preictal patterns, the NeuroPace system triggers stimulation through its eight channels, delivering pre-programmed pulses tailored to individual patient needs. These pulses consist of bursts of 200Hz for 100ms, with a pulse width of 160μs. The device adheres to a stimulation density target of $2\mu C/cm^2$ for mesial regions and $6\mu C/cm^2$ for neocortical regions, ensuring optimal therapeutic efficacy. Extensive clinical data have underscored its overall effectiveness, with a majority of patients experiencing seizure reduction. Particularly, a 9-year prospective multi-center study involving 230 patients across 34 epilepsy centers reported a median seizure reduction of 75% and a responder rate of 73%, highlighting the device's considerable therapeutic impact [27].

The NeuroPace Responsive Neurostimulator (Fig. 4(e)) stands as a pioneering example of closed-loop stimulation, widely regarded as the most prominent clinical device in this category. It employs an intracranial-EEG-based approach to seizure detection and stimulation. Implanted within the scalp, this device features both surface and depth electrodes for neural signal acquisition, and is capable of recording electrocortical data from four channels, allowing for monitoring brain activity with spatial information.

Despite its demonstrated promising efficacy in reducing seizure frequency, its reliance on rudimentary seizure detection algorithms may lead to false positives, resulting in excessive stimulation events. The device have been shown to administer thousands of stimulations daily, in the absence of behaviorally-evident or EEG-evident seizures, underscoring the significant room for improvement in this domain [28].

1.4 Conventional vs Adaptive Closed-Loop Stimulation

In conventional closed-loop systems, as shown in Figure 1-3, stimulation is triggered upon detection of a seizure, regardless of the detection algorithm type or implementation. This involves injecting electrical pulses to the brain tissue, using electrodes that could be the same or different from recording electrodes. A wide range of voltage-mode, current mode, and charge-mode stimulators have been reported in literature [29-35].

Thanks to their simple and straightforward design, and more importantly, their precise control on the charges flow to the brain tissue, current-mode stimulators are generally favorable compared to their counterparts. The charge control is both important for ensuring (i) charge balance at the electrodetissue interface [36-39], and (ii) charge injection spatiotemporal density, expressed in μC/cm2/phase, is below safety limits . These advantages come at cost of certain challenges in the design of current-mode stimulators, including issues with their energy efficiency as well as voltage compliance, requiring adaptive architectures to ensure efficiency and compliance. Such discussions are outside of the scope of this thesis and we refer readers to references discussing such issues in detail [40-42].

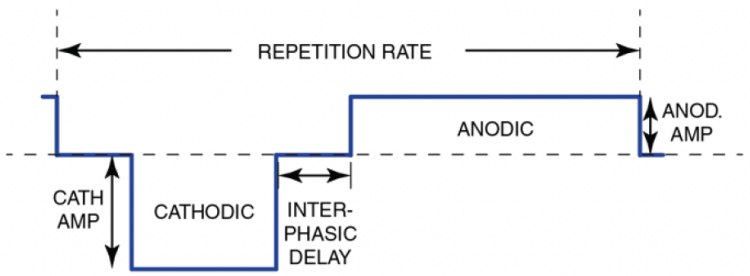

Figure 1-6 Parameters of a typical biphasic pulse use in current-mode stimulation.

Table 1-1 Commonly used ranges for biphasic stimulation pulse parameters in seizure suppression.

Parameter	Common Range
Amplitude	0.1 mA – 5 mA
Cathodic/ Anodic Phase width	60 μs – 500 μs
Repetition Rate (Frequency)	1 Hz – 200 Hz
Inter-Phasic Delay	0 μs – 100 μs
Burst Duration	100 ms – 2 s
Inter-burst Interval	1 s – 5 s
Pulse Train Duration	10 s – 5 min

For pulses delivered to each electrode, several parameters, such as magnitude, frequency, duty cycle, and shape, need to be independently set. Figure 1-6 shows a typical bi-phasic stimulation pulse, with parameters' typical values listed in Table 1-1. In conventional closed-loop stimulators, these parameters are set using a simple set-and-go strategy. Physicians conduct patient interviews, typically every 3 to 6 months, to adjust these parameters based on the seizure occurrence rate (based on patient's subjective diary) and pre-programmed sets of parameters, and the same stimulation waveform will be applied to the patient's brain for until the next appoinment.

1.4.1 Problems with Conventional Closed-Loop Stimulation

The use of the same stimulation waveform with fixed parameters and electrode for every stimulation event is a significant limitation. This approach fails to account for the temporally dynamic nature of brain activity and individual patient responses. Additionally, it overlooks potential variations in seizure characteristics and patterns over time, as well as differences in patient conditions. Applying stimulation to the same location in the brain without considering potential changes in neural circuitry or optimal target regions further exacerbates this issue. Seasonal interviews conducted by physicians to adjust parameters may miss seizures that occur during sleep or in pediatric patients. Furthermore, since seizures happen infrequently, adjusting parameters by physicians can take a long time to optimize, delaying the achievement of effective treatment outcomes. The current clinical practice also

overlooks the potential for personalized treatment optimization. This motivates the need for patient-optimized responsive stimulation.

1.4.2 Stimulation Parameter Optimization Challenges

Considering the wide range of values each of these parameters could take and a typical electrode arrays size of stimulation sites, the parameter space for stimulation is extremely large. This means that sweeping through all possibilities to find the most effective one is practically impossible, especially considering the significant time (e.g., weeks to months) that is currently required to evaluate the effectiveness of a set of parameters on a patient. Additionally, despite great advancements, the complete mechanism by which stimulation exerts its therapeutic effect for epilepsy is unknown [43] and there is no universal brain model that can accurately predict the response to different stimulation strategies across different patients and over time.

On the other hand, data-driven methods (e.g., ML-based algorithms) cannot be used due to the lack of necessary datasets for training them. To create such datasets, well-designed largescale multi-center clinical studies with long-term follow-up are required to be conducted exploring different stimulation parameters and strategies including various intensity, duration, and frequency of stimulation as well as number of sessions and stimulation targets. Additionally, the evaluation should include patients with various seizure frequency, seizure type and severity, and lost cognitive functions.

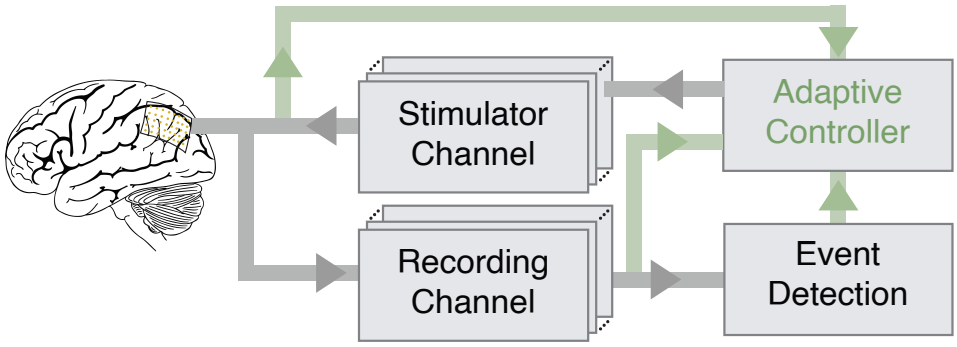

Figure 1-7 patient-optimized time-adaptive stimulation.

1.4.3 Adaptive Closed-Loop Stimulation

The above motivates for an online patient- and time-adaptive control strategy that works in a closed-loop fashion and not only triggers stimulation upon detection of a neurological event (i.e., the conventional closed-loop neuro-stimulation, shown in Figure 1-3) but also continuously adjusts the stimulation parameters based on the results of previous stimulation, as shown in Figure 1-7 [44].

Adaptive stimulation is essentially done by conceptualizing neurostimulation as a closed-loop state-space control issue, and its main objective is to develop an optimal control strategy to transition the current system state (e.g., seizure) to the target state (e.g., normal). Such an adaptive scheme has been investigated with promising results for movement disorders [45-46]. Developing this strategy for epilepsy, however, comes with some challenges.

The primary challenge lies in the inherent non-determinism of in Seizure Control. Seizure dynamics involve highly complex and nonlinear processes within the brain's network, which remains incompletely understood [47-49]. This unpredictability complicates the development of effective strategies for seizure suppression, requiring a nuanced grasp of the underlying neural mechanisms and adaptive algorithms tailored to these complexities. This stands in contrast to adaptive stimulation for Parkinson's Disease (PD), where the pathological neural patterns are more deterministic and well-characterized, making the implementation of adaptive closed-loop systems more feasible. In PD, the motor symptoms are directly linked to specific, abnormal neural oscillations that can be more consistently monitored and modulated. As a result, adaptive stimulation for PD is not only more straightforward but also already proven to be effective in clinical settings, highlighting the significant challenges unique to epilepsy.

Another significant challenge is the lack of indicators of stimulation efficacy in suppressing or preventing seizures that can be evaluated immediately after each pulse and are robust against stimulation artifacts [50-53]. For example, in Parkinson's disease, the efficacy of stimulation is apparent when the tremor stops, indicating that the stimulation is optimal. Such clear indicators are not available for epilepsy, making it difficult to assess and adjust stimulation parameters effectively.

1.5 Thesis Objective

The primary objective of this project is to design and develop an adaptive controller for a closed-loop implantable neurostimulation device, aimed at optimizing seizure suppression through real-time, patient-specific adjustments of stimulation parameters. The controller will work in tandem with a seizure detection block, identifying seizure onset and triggering the adaptive response by dynamically adjusting parameters such as amplitude and frequency based on continuous feedback from intracranial electroencephalography (iEEG) recordings.

The adaptive controller will employ online learning techniques, continuously refining its strategies based on the patient's brain activity and responses to previous stimulations. This approach allows the system to tailor interventions to each patient's unique and evolving epilepsy profile, ensuring sustained efficacy by adapting to both immediate changes during seizures and long-term shifts in the patient's neurological condition.

A key aspect of this design is ensuring that the adaptive controller operates within the stringent power budget required for implantable devices, specifically targeting a power consumption of less than 1 mW [54]. This low power consumption is crucial for the longevity and safety of the implantable device, ensuring that it can function effectively over extended periods without requiring frequent recharging or replacement.

1.6 Proposed Solution

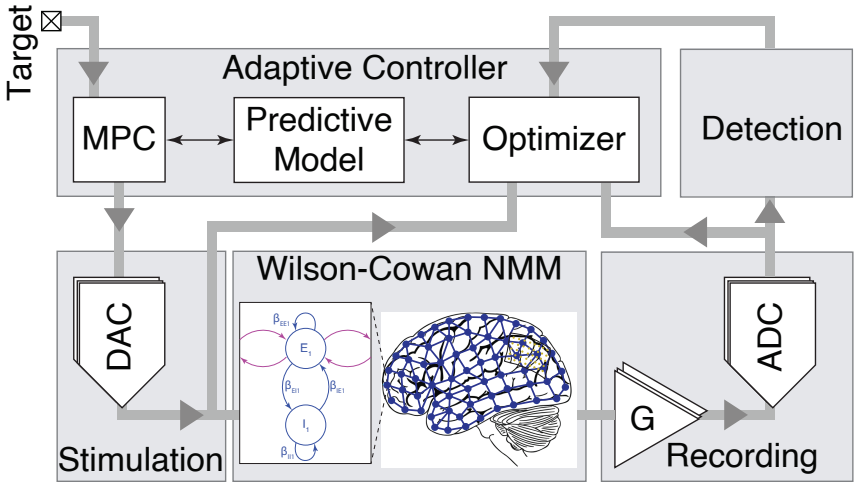

Figure 1-8 Proposed framework for training and optimization of and adaptive closed-loop neurostimulation using neural mass model

Figure 1-8 shows the top-level block diagram of the framework proposed in this thesis for developing and optimizing an adaptive neurostimulation system. This framework uses a Neural Mass Model (NMM)—a computational model designed to generate intracranial EEG (iEEG) signals with characteristics similar to those recorded from each patient. The NMM serves as a virtual brain and generates realistic patient-specific iEEG signals during both normal and seizure periods and simulates the brain's response to stimulation pulses with varying parameters. Since we use the NMM, the onset of seizures is already known, eliminating the need to develop a seizure detection block to process the iEEG signals for detection. Consequently, in our framework, the seizure detection block is a simple component that merely sends labels indicating whether we are in a normal episode or experiencing a seizure.

These synthetic iEEG signals are processed by an adaptive controller, as shown in the figure. The signals are fed into a predictive model responsible for forecasting the outcomes of different stimulation scenarios, specifically for the current brain state, to assess their efficacy. The controller continuously interacts with the predictive model to determine the most optimal stimulation parameters based on these predictions. Once identified, these parameters are sent to the stimulator, which applies the stimulation to the NMM, representing the substitute brain. An optimizer block continuously evaluates the performance of the predictive model by processing both the selected parameters and the NMM's responses, working to improve its accuracy over time and enabling ongoing learning and refinement of the system.

1.7 Thesis Overview

In Chapter 2, we will focus on the development of a NMM that is tailored to simulate the brain's electrical activity under both normal and seizure conditions. The model is rigorously validated to ensure it accurately reflects the dynamics of epileptic seizures and can realistically generate iEEG signals corresponding to different states. Additionally, we demonstrate the NMM's responsiveness to a wide range of stimulation parameters, showing that it can replicate various physiological responses to neurostimulation. This capability positions the NMM as a robust framework for exploring the parameter space, enabling the identification of stimulation strategies that are likely to be effective in seizure control. By

proving its adaptability and precision, we establish the NMM as a critical tool for the next step of our research into optimizing neurostimulation therapies.

In Chapter 3, we will introduce the Deep Extended Dynamic Mode Decomposition (Deep EDMD)-MPC framework for real-time closed-loop electrical neuromodulation in epilepsy. This framework builds on the NMM developed in Chapter 2, which provided a validated simulation environment for epileptic brain activity. It combines a deep Koopman operator-based dynamical model, which predicts the temporal evolution of epileptic iEEG signals using an approximate finite-dimensional linear dynamics, with a model predictive control (MPC) module that designs optimal seizure suppression strategies. The Koopman operator-based linear dynamical model is embedded within the latent state space of an autoencoder neural network, allowing for the online approximation and updating of the Koopman operator. The adaptive controller is implemented on an FPGA using VHDL, ensuring real-time performance and integration with the neuromodulation system.

In Chapter 4, we conclude our work and outline potential avenues for future research.

1.8 Thesis Contributions

The research contributions of this thesis have led to the following publications, which include a first-authored conference paper presented at IEEE ISCAS 2024 (Chapter 2) and a co-

authored journal paper published in IEEE TBioCAS 2024. The content of Chapter 3 has not yet been published.

1. R. Salahi, H. Kassiri, “NMM-Based Patient-Specific Temporally-Adaptive Stimulation Optimization for Seizure Control.” In 2024 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5. IEEE, 2024.
2. H. Kassiri, A. Muneeb, R. Salahi, and A. Dabbaghian, “Closed-Loop Implantable Neurostimulators for Individualized Treatment of Intractable Epilepsy: A Review of Recent Developments, Ongoing Challenges, and Future Opportunities,”, Accepted (Early Access), IEEE Transactions on Biomedical Circuits and Systems

Chapter 2 Patient-Specific NMM: A Computational for the Development of Adaptive Neurostimulation Controllers

2.1 Introduction

As described in the previous chapter (Section 1.6.), the envisaged adaptive controller requires a block that predicts the outcome of stimulating the brain with a specific set of pulse parameters. To improve the prediction accuracy, the adaptive controller needs to be continuously trained, by taking the stimulation pulse parameters (i.e., input to the brain) and the resulting neuronal activity (i.e., brain's response) as its inputs. However, there are certain challenges to consider: First, the adaptive controller's training can only be done during seizure episodes when stimulation is applied, resulting in new training data points; second, seizures are inherently sparse in time, occurring, for example, once a day and lasting for only tens of seconds; and third, the stimulation parameter space is extremely large, as previously discussed.

Due to these factors, the adaptive controller's training cannot be performed entirely online on a patient's brain, as it would require an unreasonable amount of time, possibly spanning years, to achieve acceptable prediction accuracy. On the other hand, offline training using pre-recorded data is not viable, as there is currently no dataset containing the neuronal responses of a diverse range of patients to all possible combinations of stimulation pulse parameters.

These constraints motivate the use of a numerical neural dynamics model (e.g., neural mass models (NMM)) as a substitute for the brain during the training process and for testing the designed adaptive controller. This model, as a substitute for the patient's brain, must meet certain requirements to be a good replacement. It needs to accurately replicate the neuronal dynamics of the human brain, particularly during seizure episodes, capture the variability in neuronal responses to different stimulation pulse parameters, and allow for the generation of sufficient training data within a reasonable timeframe.

It is important to emphasize that the neural mass model (NMM) we use is not intended to perfectly replicate or replace the neuronal dynamics of any specific patient. Instead, the NMM serves as a computational testbench or playground, providing a controlled environment in which we can develop and evaluate our adaptive controller. The goal is to ensure that the model generates EEG signals that are as realistic as possible, particularly in distinguishing between normal and seizure states, and in producing varied responses to different stimulation waveforms and pulse parameters. By carefully tuning the NMM parameters, we aim to create

a versatile and adaptable model that mimics brain-like activity in a way that is generalizable across different patients. This approach allows us to validate the adaptive controller in a simulated setting before any patient-specific optimization or clinical application, recognizing that the optimal solutions derived from the NMM may not directly translate to those required for a real patient's brain but will guide us toward a robust and effective framework for adaptive stimulation.

2.2 NMMs Background

The Hodgkin–Huxley model, revealed how spikes in single neurons arise from the interplay of depolarizing and hyperpolarizing currents [55]. However, complex behaviors like movement and perception emerge not from individual neuron spikes but from the collective action of large networks of neurons [56-57]. Techniques like macroscopic functional imaging techniques such as functional magnetic resonance imaging (fMRI) and electroencephalography (EEG) provide insights into the collective activity patterns of thousands of neurons [58], motivating the development of mean field approaches, which shift focus from individual neurons to populations of neurons. These models have been successful in various domains, including modeling seizures [59], encephalopathies [60-61], sleep [62], anesthesia [63], resting-state brain networks [64-65], and the human alpha rhythm [66-67].

Despite the nonlinear nature of single-cell spikes, at large scales, neural ensemble activity can often be approximated using simpler, linear statistics like mean and variance. The dynamics of a linear, normally distributed neural ensemble can be described by the Fokker-Planck equation (FPE). This equation can be derived analytically from simple single-neuron models, such as the integrate-and-fire model, assuming the diffusion approximation holds [68-72]. the FPE describes the population variance, indicating the precision of the ensemble's response [73-74]. As the inputs to the neural ensemble change, the FPE accounts for the drift in the mean and the diffusion, or changes in variance, of the ensemble activity. It accommodates individual nonlinearities, local correlations between neurons, and variations within neuron families through the diffusion approximation.

In the presence of strong coherence, it may be reasonable to assume that the ensemble activity is close enough to the mean that the variance can be disregarded. This assumption reduces the dimensionality to one, enabling the modeling of multiple interacting local populations, such as excitatory and inhibitory neurons in different cortical layers, with a small set of equations describing the mean activity of each neural population [75-76]. This mass-action approach is fundamental to neural mass models (NMMs) [77].

One method for constructing NMMs follows the empirical approach of Hodgkin and Huxley. This method acknowledges that complex systems can exhibit specific behaviors at different organizational levels, suggesting that large-scale activity may be more than just the

sum of its parts. Notable empirically informed NMMs include the Wilson–Cowan [78] and Jansen–Rit [75] models. Additionally, there are hybrid methods combining theoretical population dynamics with empirical synaptic and input-response functions [79-81].

2.3 NMM Development

In this work, we have chosen the Wilson-Cowan NMM (WC-NMM) and fine-tuned its parameters based on each patient's pre-recorded EEG data (details in the following sections). This fine-tuning ensures that the WC-NMM output accurately emulates the patient's macroscopic neuronal behavior during both normal and seizure periods. The selection of the WC-NMM is based on several key features common to various neural mass models: (i) its ability to model the response of neuronal populations to external inputs, including electrical stimulation, with adjustable amplitude, frequency, etc; (ii) its capacity to incorporate nonlinear interactions between excitatory and inhibitory populations, enabling the capture of a wide range of neural dynamics, including oscillations and pattern formation; and (iii) its focus on the interaction between excitatory and inhibitory sub-populations, rather than on cellular-level neuronal dynamics [82]. Additionally, the WC-NMM was chosen for its relatively limited number of parameters, which allows for more straightforward modeling, and its similarity to other established models, making it a suitable choice for our study.

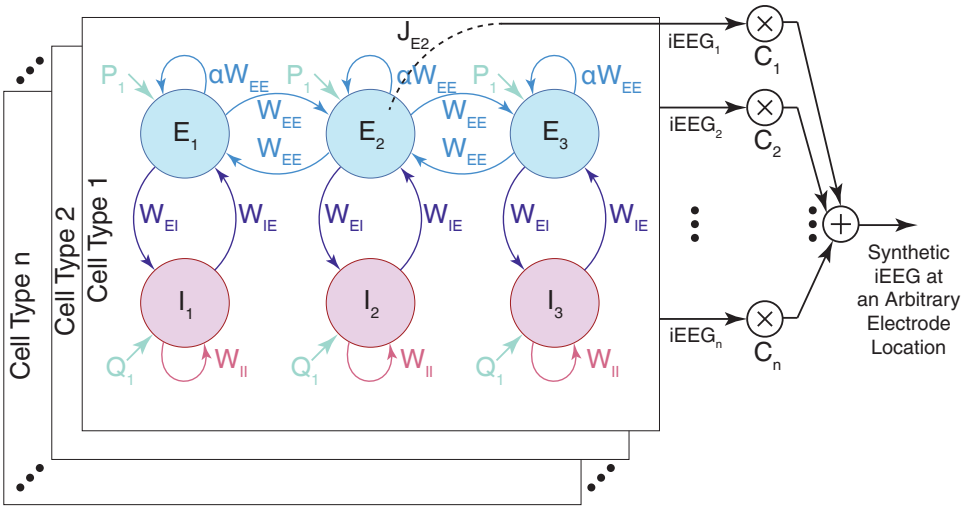

Figure 2-1 Simplified representation of a WC-NMM with n cell types, each three populations, and the spatial averaging used for iEEG output

Figure 2-1 illustrates the design of the WC-NMM employed in this study featuring n different cell types (n=50 in this work), each including a set of three populations comprising both excitatory (E) and inhibitory (I) sub-populations. This approach was taken because intracranial EEG signals essentially represent a spatial average of electrical activity from various types of neurons near the recording electrode. Therefore, to accurately mimic these diverse neural activities, we designed an interconnected NMM with different types of cell populations, each characterized by distinct parameters. This approach ensures that the synthetic iEEG generated by the NMM closely resembles real iEEG signals and captures the unique activity patterns of different neural types (simulation results confirming this are presented in the following section). The WC-NMM is governed by the following equations:

$$\tau_E \frac{\partial E_k(t)}{\partial t} = -E_k(t) + (1 - E_k(t)) F_E\left(J_{E_k(t)}\right) \quad (1\text{-}1)$$

$$\tau_I \frac{\partial I_k(t)}{\partial t} = -I_k(t) + (1 - I_k(t))F_I(J_{I_k(t)}) \quad (1\text{-}2)$$

$$J_{E_{1,3}(t)} = W_{EE}E_{1,3}(t) - W_{IE}I_{1,3}(t) + \alpha W_{EE}E_2(t) + P_{1,3}(t) \quad (1\text{-}3)$$

$$J_{E_2(t)} = W_{EE}E_2(t) - W_{IE}I_2(t) + \alpha W_{EE}(E_1(t) + E_3(t)) + P_2(t) \quad (1\text{-}4)$$

$$J_{I_{1,2,3}(t)} = W_{EI}E_{1,2,3}(t) - W_{II}I_{1,2,3}(t) + Q_{1,2,3}(t) + s(t) \quad (1\text{-}5)$$

Here, E and I represent the proportion of excitatory and inhibitory cells firing per unit time at the instant t. τ_E and τ_I represent the time constants for the E and I sub-populations, respectively. $F_{E/I}$ corresponds to the sigmoid activation function with threshold values $\theta_{E/I}$ and scaling factor $A_{E/I}$, which determines the firing rate of the neural populations. $J_{E_{1,2,3}}$and $J_{I_{1,2,3}}$ represent the input currents to the excitatory and inhibitory populations, respectively. P and Q represent external inputs from other cell type neurons to the respective populations, S(t) represents the stimulation, and the W coefficients denote the strengths of connections as indicated by their indices: W_{EE} represents the strength of the excitatory-to-excitatory connection, α is a scaling factor, W_{EI} represents the strength of the excitatory-to-inhibitory connection, W_{IE} represents the strength of the inhibitory-to-excitatory connection, and W_{II} represents the strength of the inhibitory-to-inhibitory connection. The initial values for these parameters are set based on findings from previous modeling studies (e.g., [78], [83-87]). Table 2-1 shows the initial values in our model.

Table 2-1 Initial values of the NMM parameters.

Parameter	Value
τ_E and τ_I	1-50 (ms)
θ_E	5.25
θ_I	3.75
A_E	1.58
A_I	2.22
W_{EE}	16
W_{EI}	18
W_{IE}	12
W_{II}	3
P, Q	1
α	0.1

As shown in Figure 2-1, the synthetic iEEG created by cell type k (i.e., $iEEG_k$) is defined as the input current of its second population, chosen for the sake of consistency. The figure also shows that the iEEG recorded from a virtual electrode, placed at an arbitrary point of the space modeled by the NMM, is the weighted spatial average of $iEEG_1$ to $iEEG_n$, with weights (C_1 to C_n) being inversely proportional to the distance between the electrode and the cell types, i.e., higher weight value for closer cells. These weights and some of the NMM's network-level parameters (e.g., P, Q, and α) are later used to fit the synthetic iEEG created by the NMM to the real iEEG signals recorded from different electrodes.

2.3.1 Python Implementation

To numerically solve the WC-NMM, the first step involved converting the original set of equations into a system of first-order stochastic differential equations (SDEs). The SDEs were

solved using the Euler-Maruyama method [88], which discretizes time into small intervals and approximates the continuous-time stochastic process using a sequence of random variables. Additionally, each subpopulation of each cell type receives an independent noise input denoted as μ(t), which is added to the Wilson-Cowan equation for that subpopulation. This noise reflects the inherent stochastic fluctuations that accompany the activity of a population of neurons in the brain. These fluctuations arise from factors such as thermal energy, ion channel chattering, and irregular synaptic inputs from other neurons [89]. For the presented NMM, this noise input is derived from an Ornstein-Uhlenbeck process [90], described as:

$$\frac{d\mu}{dt} = -\frac{\mu^{ext} - \mu}{\tau_{ou}} + \sigma_{ou}\zeta(t) \tag{1-6}$$

in which, μ^{ext} denotes the mean of the process and serves as a constant external input. The parameter τ_{ou} governs the time scale, while $\zeta(t)$ represents a white noise process derived from a normal distribution with a mean of zero and unit variance. The magnitude of the fluctuations around the mean is determined by the noise strength parameter σ_{ou} [83] . The model parameters were initialized with biologically-motivated values [78], [83-87]. A time step of ~976μs was used for simulations, corresponding to the sampling rate of 1024Hz used for our pre-recorded datasets [91].

2.3.2 Patient-Specific Optimization

As discussed, customizing the NMM for each patient involves fine-tuning its parameters to closely match the NMM-generated synthetic iEEG ($iEEG_{syn}$) with the real prerecorded iEEG ($iEEG_{real}$) of that specific patient. In this work, the source of our pre-recorded iEEG data is the SWEC-ETHZ dataset, which contains recordings from 18 patients, totaling 2656 hours, and encompasses 116 seizure episodes [91]. To evaluate how well the synthetic and real iEEGs are matched, we opted for spectral similarity over temporal one. This was done because our analysis of different epochs of prerecorded $iEEG_{real}$ revealed that a strong time-domain correlation is only present when comparing simultaneous signals from neighboring channels, which is expected given their spatial proximity. However, such correlation did not manifest when comparing randomly-selected $iEEG_{real}$ epochs. In contrast, our frequency-domain analysis revealed consistent high levels of correlation (e.g. 0.6-0.9) in randomly-selected epochs of the same size, regardless of their spatial proximity.

Our NMM optimization strategy followed a two-phase approach. Prior to optimization, we initialized all its parameters with biologically-plausible values obtained from literature [78], [83-87]. During the first phase, we focused on matching the output for normal (i.e., non-seizure) episodes, and fine-tuned NMM parameters to achieve that. This included parameters associated with the distance of cell types from the electrode (i.e., C_1 to C_n).

The optimization process utilized the Nelder-Mead algorithm [92], using 10-second $iEEG_{real}$ episodes as the input. The resulting parameters underwent further refinement through averaging to establish the finalized set of parameters. To ensure a comprehensive coverage of diverse brain dynamics, the optimization process extended to numerous randomly-selected normal epochs. This extensive sampling allowed for capturing variable dynamics inherent in different brain states, which contributed to a more accurate representation of the patient's neuro-electrical activity.

To evaluate the spectral similarity of the ($iEEG_{syn}$ NMM's output) against $iEEG_{real}$, we calculated the point-by-point squared difference between the normalized power spectral density (PSD) of the $iEEG_{syn}$ and $iEEG_{real}$ and compared that to an acceptable threshold value. The threshold value was established by taking the average error value across a statistically-significant number of randomly-selected, diverse $iEEG_{real}$ epochs from different channels and with varying durations (1, 5, 10, and 30 seconds).

Given that seizure onset is abrupt and seizure dynamics often exhibit oscillatory characteristics, it is noteworthy that seizures can be considered a form of oscillatory activity, although not all oscillations resemble seizures. This similarity in dynamics aligns with some experimentally observed features of seizure-like activity [82], [93-96].

During the second phase, to generate seizures in $iEEG_{syn}$ with similar spectral characteristics as seizures in $iEEG_{real}$, we fine-tuned parameters P, Q, and α for certain cell

types to shift their normal behavior towards seizure-associated oscillatory patterns. These specific cell types were selected based on their natural oscillation frequency that is set by their τ_E and τ_I, which allowed for generating an oscillatory behaviour matched to the oscillation frequency of $iEEG_{real}$ during a specific seizure.

2.4 NMM Performance Evaluation

NMM performance evaluation was conducted for both phases to ensure accuracy and consistency. Figure 2-2(a) shows two time-domain examples of 10-second $iEEG_{real}$ epochs from a specific patient, compared to $iEEG_{syn}$ generated by a WC-NMM that its parameters were optimized for that patient. Figure 2-2(b) shows the PSD of the two examples and how matched the spectral characteristics of the NMM's $iEEG_{syn}$ is compared to $iEEG_{real}$. To quantify this similarity, we measured the previously-defined spectral error between the two PSDs while shifting the 10-second window by 1 second for 5 seconds.

As shown in Figure 2-2(c), the error is consistently below the targeted threshold of 41.5μV^2/Hz, which is calculated as discussed in the previous section. Additionally, figure 2-2(d) compares the PSD of the patient-optimized $iEEG_{syn}$ to the $iEEG_{real}$'s average PSD$\pm$standard deviation as a statistically significant evidence illustrating the NMM's capability in generating spectrally-similar patient-specific iEEG signals. These results were repeated for data from

different channels and patients, confirming the efficacy of the first phase of our two phase optimization method.

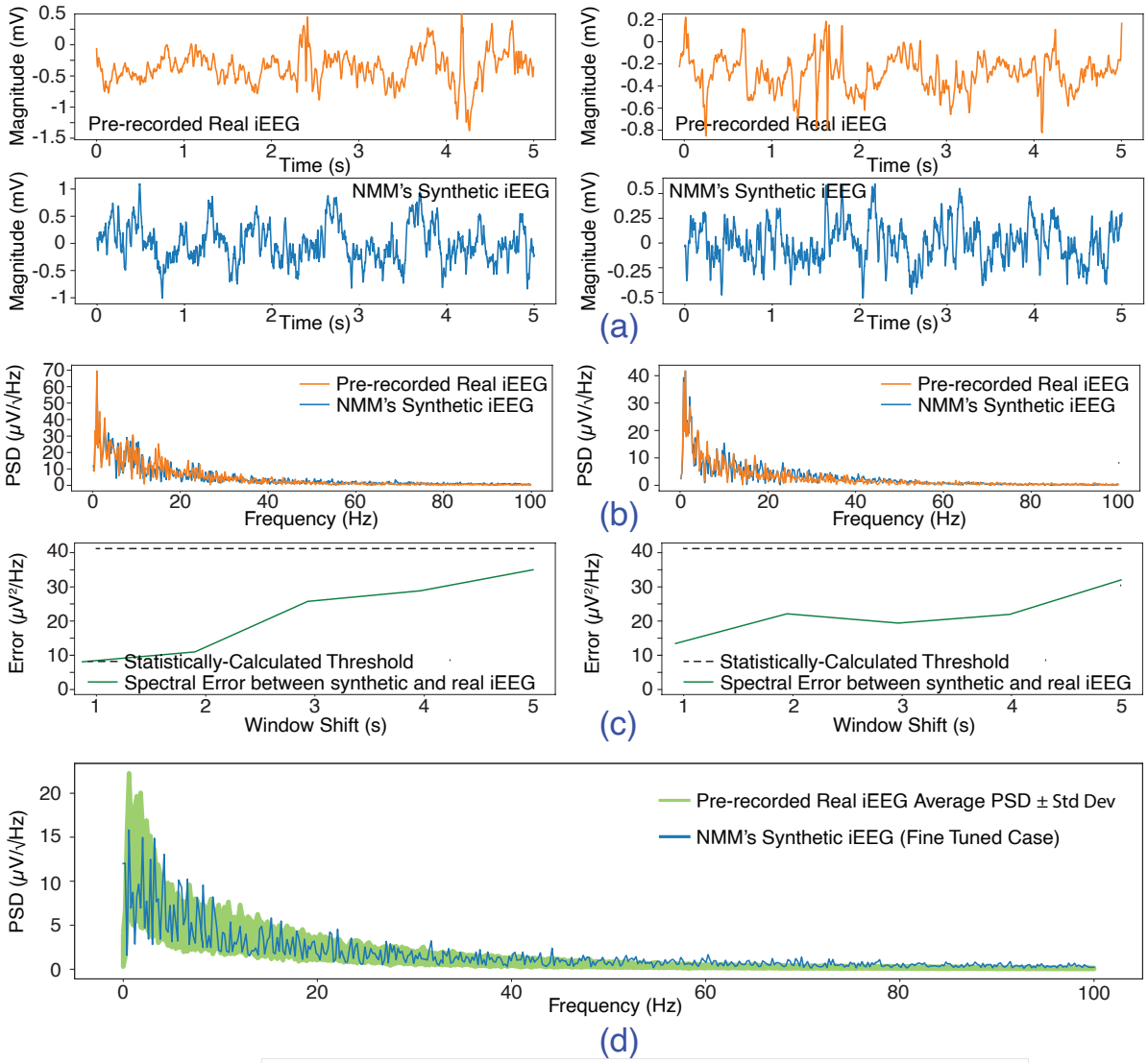

Figure 2-2 Comparison of two example of synthetic iEEG and real iEEG

To validate the second phase of optimization (i.e., generating patient-specific seizure-like behaviour using NMM), we first investigated spectral characteristics of real pre-recorded iEEG signals during a seizure event. An example is shown in Figure 2-3, in which normal

iEEG$_{real}$ and seizure iEEG$_{real}$ are compared in both time and frequency domains, which demonstrate a clear shift in signal's spectral energy during seizures.

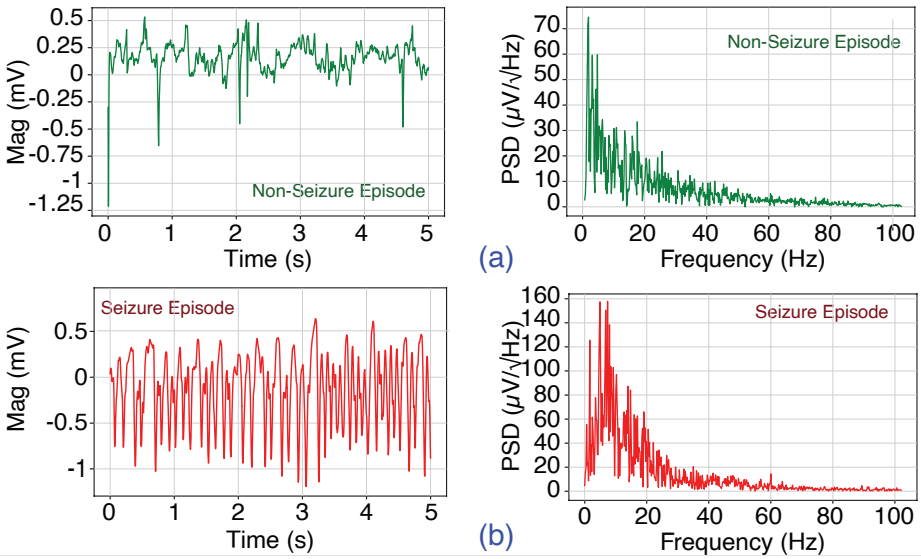

Figure 2-3 Comparison of time and frequency domain of real iEEG during normal and seizure periods

Upon further investigation, we observed oscillations in the shifting of energy to various frequencies at different times during a patient's seizure. While this is just an example, and the spectral change varies across different patients, certain oscillatory patterns have been observed consistently for the same patient over time. As discussed, the second phase of our optimization method targets generating oscillatory behaviour with arbitrarily-selected dominant frequencies. Figure 2-4(a) shows the synthetic iEEG generated by the NMM, in which P and Q are adjusted for specific cell types at t=5s to generate seizure-like activity. Figure 2-4(b) and (c) show the signal's PSD for seconds 0-5 and 5-10, respectively, exhibiting similar spectral characteristics observed in iEEG$_{real}$ shown in Fig. 8, though with seizure associated oscillations at a different frequency.

Figure 2-4(d) shows a few examples of arbitrary PSDs that the NMM can generate, exhibiting its fine control on spectral distribution of its output for generating a wide range of seizure-like activities.

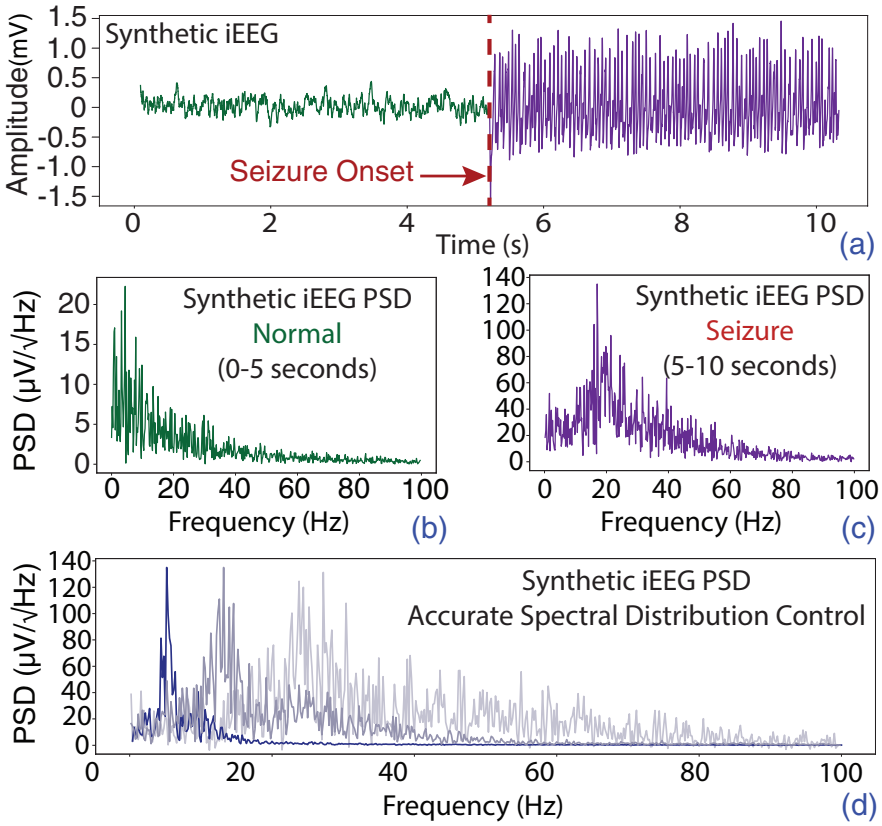

Figure 2-4 An example of NMM-generated synthetic iEEG showing the transition from a normal and a seizure state

2.5 Existence of an Optimal Stimulation

As discussed in Chapter 1, effectively suppressing seizures requires optimizing the parameters of stimulation. An important issue that has received little practical attention—though it has been extensively studied as a theoretical neuroscience research topic [97-99]—

is how to set the stimulation pulse parameters for the most effective seizure control. Since our proposed NMM will be used as a virtual substitute for the patient's brain to validate and test the proposed adaptive controller, which will be discussed in Chapter 3, this section investigates the efficacy of different stimulation parameters within our model and explores the existence of optimal stimulation conditions. To evaluate stimulation efficacy, we defined a cost function specifically for this purpose.

As discussed, seizure onset results in increased energy in certain frequencies. To evaluate stimulation efficacy, we defined the cost function as the ratio of the spectral correlation between the pre- and post- stimulation $iEEG_{syn}$ to the spectral correlation between the post-stimulation $iEEG_{syn}$ and the average PSD derived from numerous randomly-selected non-seizure $iEEG_{syn}$ epochs generated by the same NMM. In the optimization process, we aim to determine the optimal amplitude and frequency for a biphasic pulse with a 50% duty cycle that is most effective at suppressing seizure-like oscillations.

We applied different randomly selected stimulation parameters to evaluate their efficacy. Figure 2-5 compares the two sets of stimulation parameters that resulted in the highest and lowest investigated cost values, highlighting their different impacts on suppressing seizure-associated oscillations. As shown, for both examples, at t=5s, NMM's parameters are adjusted to induce a seizure, and at t=10s, stimulation is started. While the stimulation results

in amplifying the oscillatory behaviour in Figure 2-5(a), it can successfully suppress it in Figure 2-5(b), clearly illustrating the importance of stimulation parameter optimization.

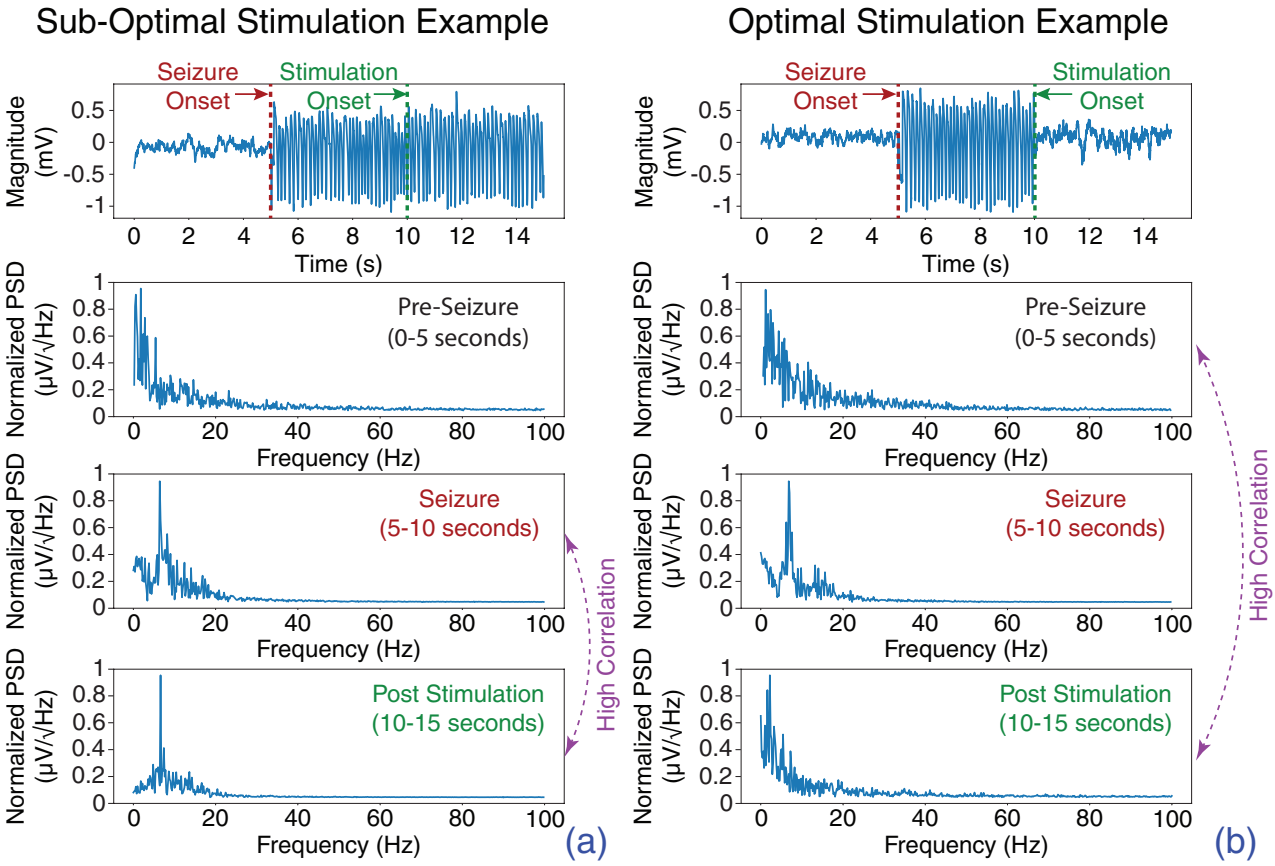

Figure 2-5 Comparison of sub-optimal and optimal stimulations applied to two seizure example of $iEEG_{syn}$, illustrating the differences between stimulation parameters in their efficacy for seizure suppression

To identify the most optimal stimulation, we swept the parameters across the parameter space (sampling 100 data points in the frequency range of 0-200 Hz and 100 data points in the amplitude range of 0-5 mA) while our NMM was exhibiting specific frequency seizure-like oscillations, and calculated the corresponding cost values. We repeated this process multiple times with the same NMM parameters, and Figure 2-6 shows the heatmap of these cost values across the parameter space in two different experimental runs.

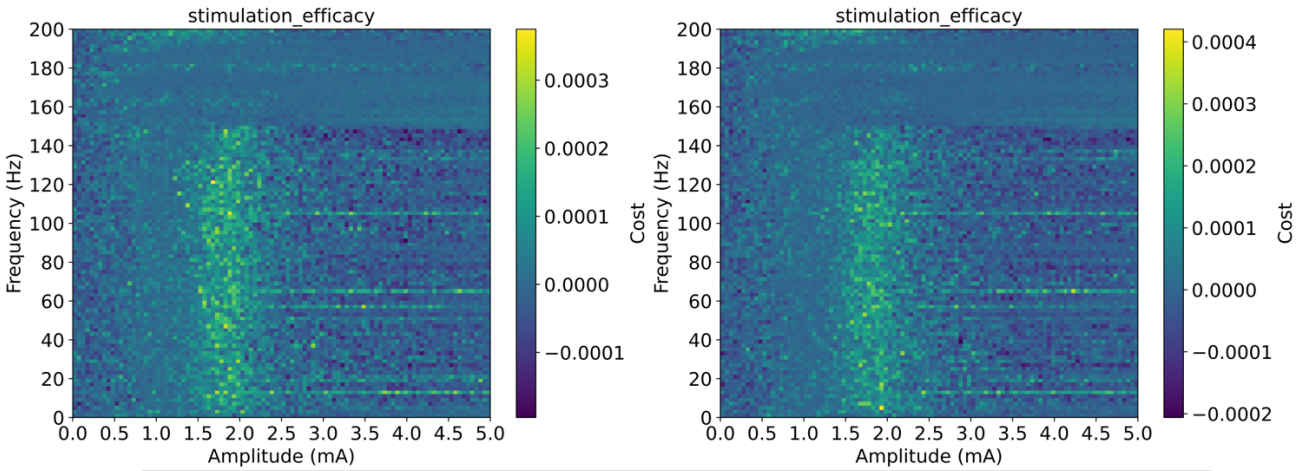

Figure 2-6 Heatmap of cost values across the parameter space in two different experimental run.

2.5.1 Using Spectral Bands Instead of Discrete Frequencies

Upon closer examination, we observed that applying the same stimulation wave to the NMM under identical conditions resulted in varying outcomes each time. After repeating this experiment five times, we found that, on average, the corresponding cost values for different stimulation parameters varied by 14%. We concluded that the presence of noise caused this variability, and thus, comparing the PSD does not reliably indicate stimulation efficacy. Consequently, we developed an alternative cost function that compares the PSD of different spectral bands rather than single integer frequencies. This new cost function also has the advantage of requiring fewer computation resources, hence, is more hardware friendly.

Numerous studies have used spectral band energies, particularly in the range from 0.1 to 30 Hz, as highly-effective features for seizure detection in real EEG data [100-104]. Figure 2-7 illustrates the Energy Spectral Density (ESD) of a one-second NMM-generated iEEG signal

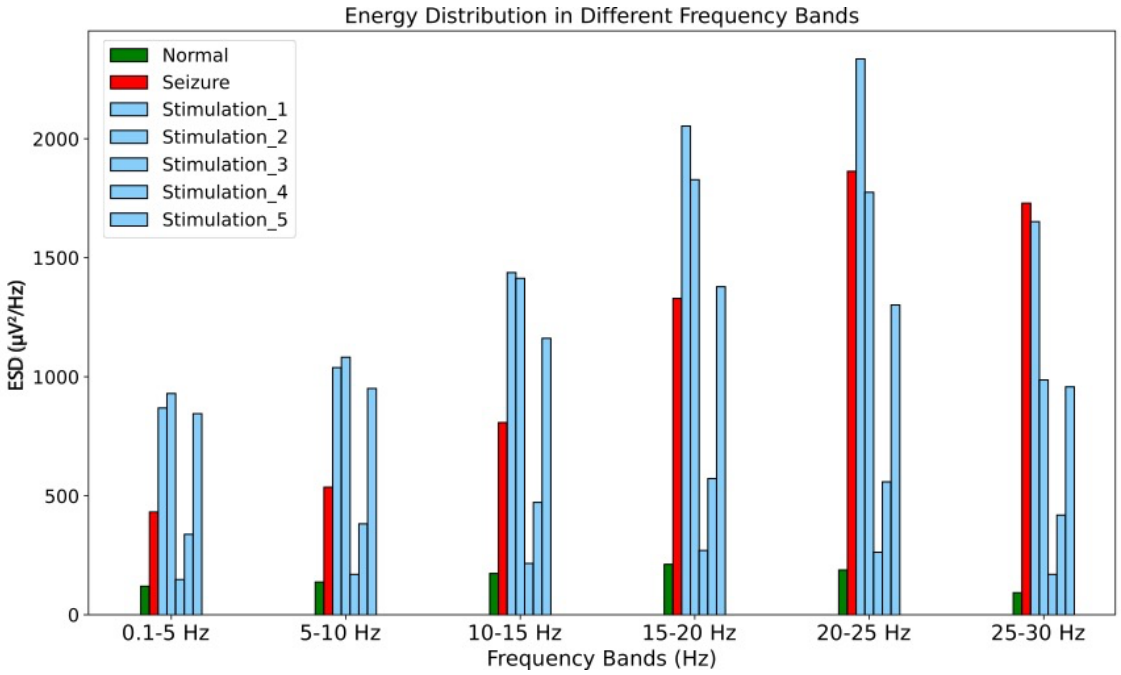

Figure 2-7 Energy of a one-second NMM-generated iEEG signal in different states: normal, during a seizure episode before stimulation, and during a seizure after applying five different square pulse stimulations.

in various states: normal, during a seizure episode before applying stimulation, and during a seizure after applying five different square pulse stimulations. As shown in figure 2-7, the ESD of the signal in some bands increases more than in other bands during a seizure. Additionally, different stimulations affect the ESD of the signal in varying ways. However, the changes in spectral band energies can vary significantly across different patients, highlighting the necessity for a patient-specific approach to effectively utilize these features in seizure suppression.

We designed the cost function to quantify the impact of stimulation on the ESD of the signal across different frequency bands. Initially, each frequency band was assigned a weight

proportional to the magnitude of its energy increase during seizure activity. This weighting reflects the relative importance of each frequency band in seizure detection and monitoring.

Next, we computed the difference in the ESD of the signal between the stimulated and normal states for each frequency band. This difference was then multiplied by the corresponding band weight. Finally, we summed the absolute values across all bands to produce a single cost value. To enhance the clarity of differences, we took the logarithm of the sum of the resulting values from all bands to produce the final cost value.

In essence, the cost function integrates these weighted differences in signal energy across relevant frequency bands. It provides a comprehensive measure of how effectively different stimulation protocols alter the spectral characteristics of the iEEG signal towards desired therapeutic outcomes and reduce the oscillatory behavior during seizure episodes.

To determine the optimal number of bands for calculating the cost values, we performed a parameter sweep using the same method described earlier, while our NMM was exhibiting specific seizure-like oscillations. We then calculated the corresponding cost values using different numbers of bands. Figure 2-8 shows the heatmap of the logarithm of cost values in the parameter space of amplitude and frequency with 50% duty cycle stimulation, where cost values were calculated using 4 bands, evenly covering the range from 0.1 to 30 Hz. Figure 2-9 presents a similar heatmap, but with cost values calculated using 8 bands. Comparing these

two heatmaps illustrates that with a greater number of bands, the efficacy of different stimulations becomes more distinguishable.

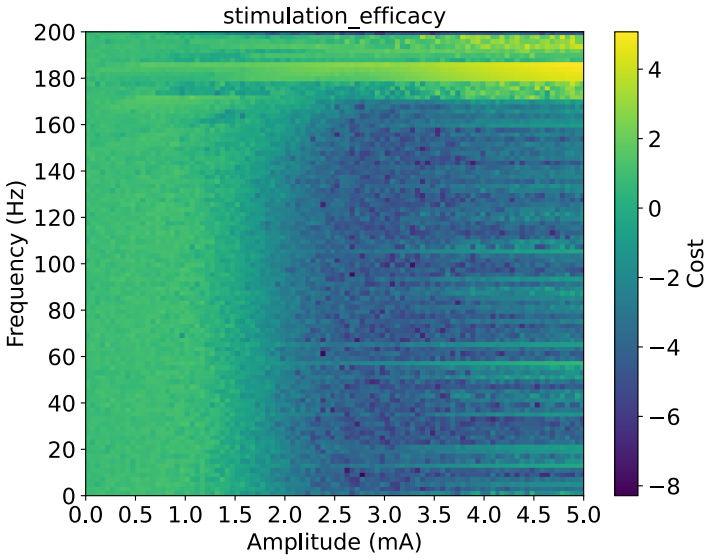

Figure 2-8 Heatmap of logarithm of cost values in the parameter space of amplitude and frequency (50% duty cycle), calculated using 4 bands.

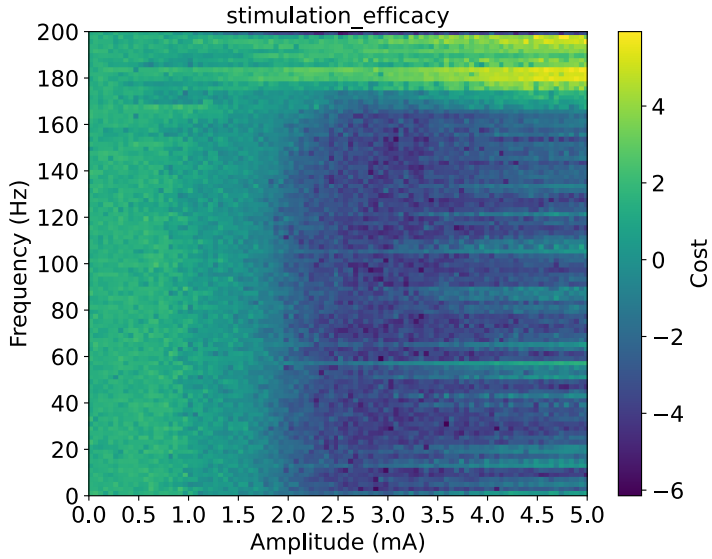

Figure 2-9 Heatmap of logarithm cost values in the parameter space of amplitude and frequency (50% duty cycle), calculated using 8 bands.

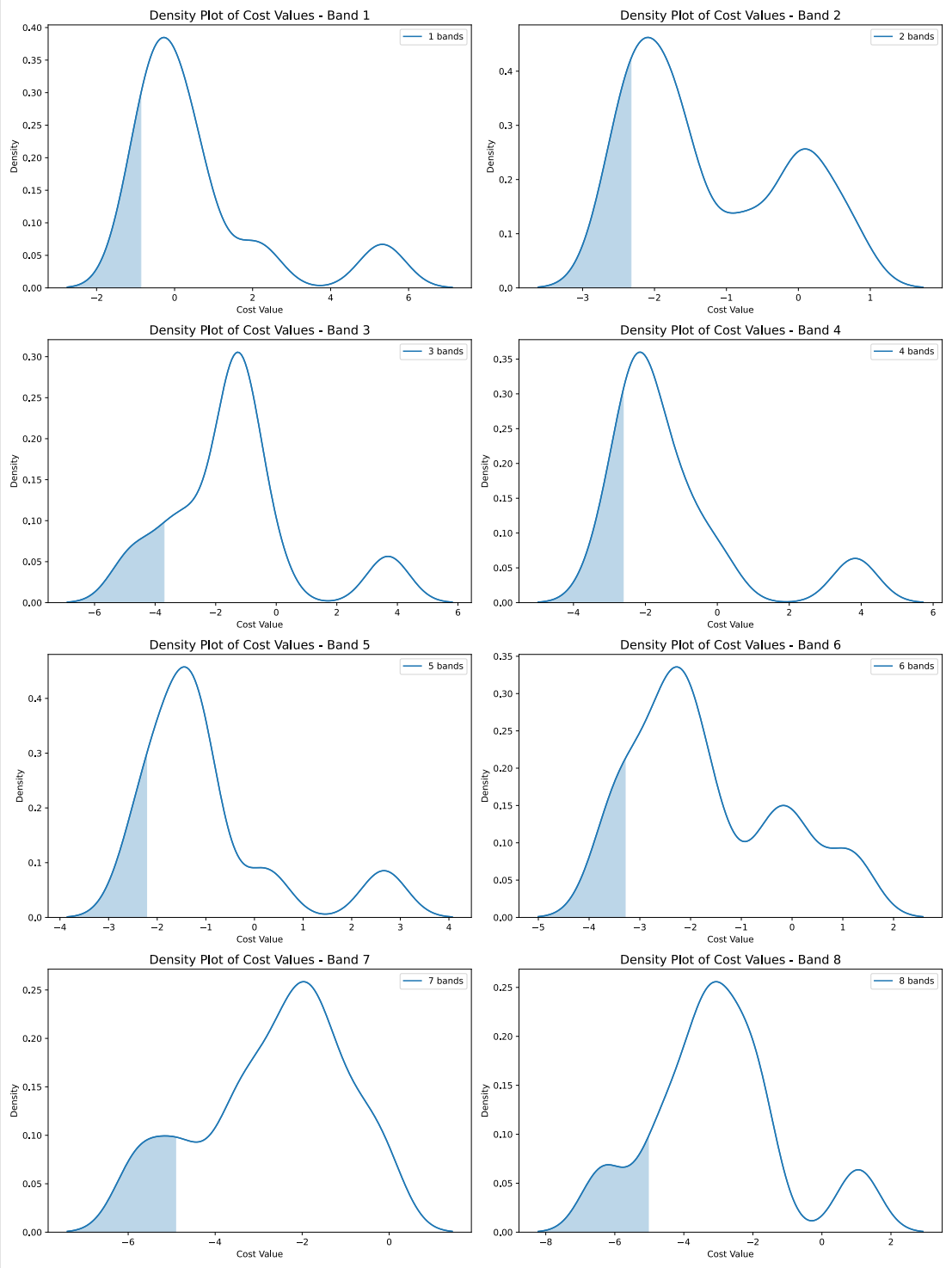

Figure 2-10 Distribution of cost values calculated with different numbers of sub-bands

To maximize the efficacy of this method, we explored various possibilities for the optimal number of sub-bands (i.e., dividing the 30Hz range by the width of each band). Figure

2-10 shows the distribution of cost values calculated for different stimulation parameters using various numbers of sub-bands, 10% of data points with lower cost values highlighted in shadow. As seen in the figure, increasing the number of bands results in 10% of data points falling within the lowest cost values and greater differentiation between these lower cost values and those with higher density.

This increased discrimination is advantageous for optimization, as it allows the controller to more effectively identify and distinguish between optimal points. However, since this cost function will later be implemented in hardware (Chapter 3), and calculating the ESD of the signal with more bands requires additional hardware resources, we also considered the trade-off between sensitivity to noise and the complexity of hardware implementation. Based on this analysis, we decided to proceed with six sub-bands, as it offers a balance between discrimination for optimization and hardware feasibility.

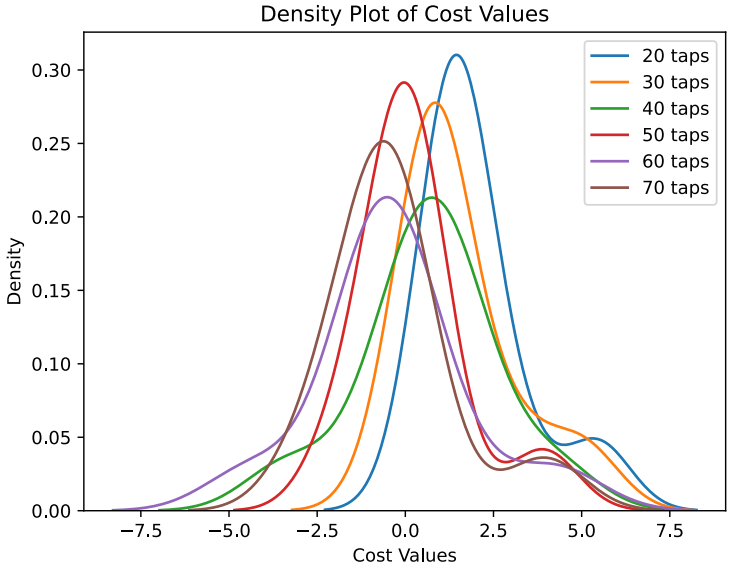

Figure 2-11 The effect of bandpass FIR filters' number of taps on the distribution of the cost value.

After determining the number of bands for calculating the cost value, we chose to calculate energy using FIR filters instead of summing the PSDs, as FIR filters require fewer hardware resources. To design the FIR filters, we calculated the cost values of stimulation parameters using FIR filters with varying numbers of taps. Figure 2-11 shows the distribution of cost values calculated with different tap numbers. Our goal was to find a lower tap count that would result in cost values with a distribution matching that of an ideal filter. Based on this analysis, we decided to proceed with 50 taps.

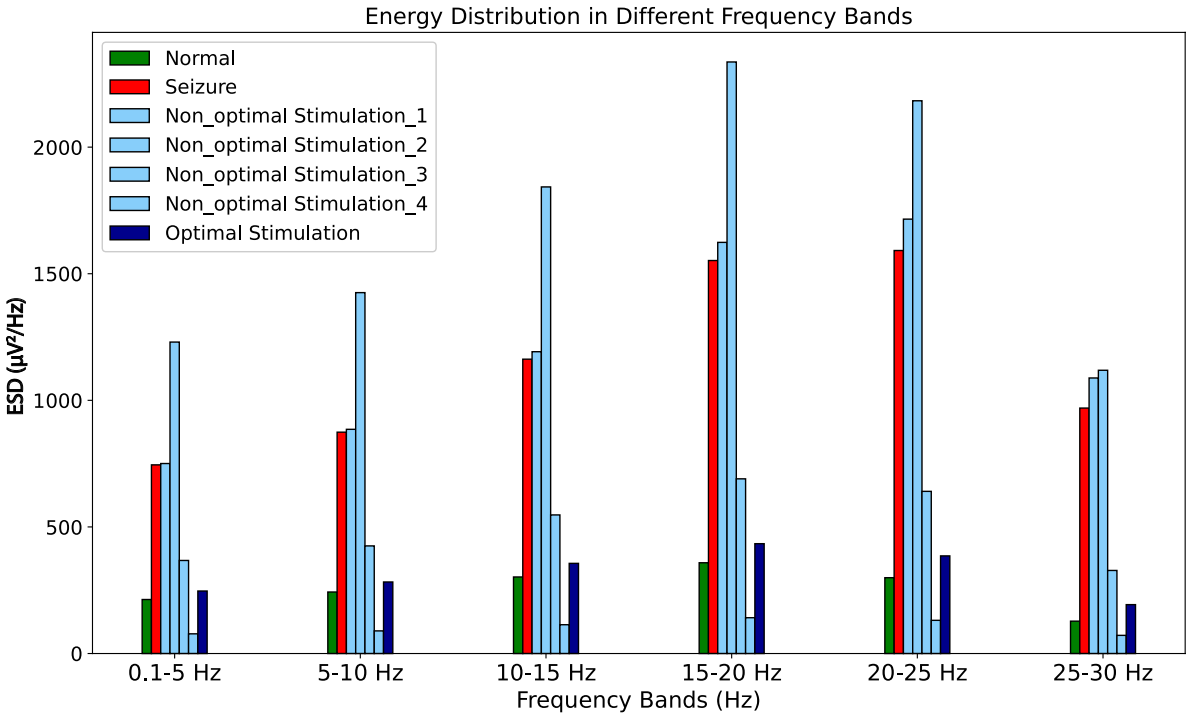

Figure 2-12 Comparison of five sets of stimulation parameters (4 non-optimal and 1 optimal) and their varying impacts on suppressing seizure-associated oscillations in energy bands.

Figure 2-12 compares the five sets of stimulation parameters and their varying impacts on suppressing seizure-associated oscillations in energy bands. It can be observed that the optimal stimulation results in a (i) reduction of energy across all bands and (ii) shows the

smallest difference in energy levels compared to normal conditions, making it more effective than the other stimulations.

The results presented in this chapter are robust and repeatable, achieved through extensive simulations across multiple scenarios. We ensured the repeatability by conducting simulations with different initial conditions and varying levels of noise, confirming that the observed patterns and outcomes were consistent. Unlike previous approaches where individual frequencies were analyzed in isolation, our method's use of weighted frequency bands provides a more reliable and comprehensive assessment of the iEEG signal's spectral characteristics under stimulation.

Furthermore, these results underscore the quality and reliability of our NMM. The model effectively captures the dynamic behaviors of seizure activity and its response to stimulation, demonstrating its appropriateness for use in developing an adaptive controller. By accurately reflecting the complex interactions between frequency bands and stimulation parameters, our NMM provides a solid foundation for creating a patient-optimized adaptive stimulation controller.

Chapter 3 Patient-Optimized Adaptive Predictive Controller

3.1 Introduction

By framing closed-loop neuromodulation as a state-space control problem, the primary objective becomes designing an optimal control strategy to guide the system's current state to a desired state [44]. Given the highly dynamic nature of neurophysiological processes, it is crucial to consider feedback mechanisms [105]. This involves leveraging the continuous flow of system measurements (i.e., iEEG recording) to tailor the control strategy to the individual's specific process. One particularly successful method in various engineering applications is model predictive control (MPC). MPC employs three key concepts [106-111]: (i) a model-based approach, (ii) the ability to predict the system's evolution and states based on a feedback control strategy aimed at optimizing an objective that incorporates risk assessment of abnormal behavior, and (iii) the receding finite-horizon re-evaluation of the control strategy's performance [112].

Developing a patient-specific MPC system for closed-loop neurostimulation to suppress seizures involves utilizing a recording signal from the patient's brain, such as EEG, to gain insights into brain activity. As shown in Figure 3-1, which presents the top block diagram of the system, these signals are processed by a predictive model designed to capture and represent the patient's brain dynamics. The model identifies the relationship between applied stimulations and the resulting changes in the EEG signals, enabling it to predict the effects of various stimulation patterns on the brain. Based on these predictions, the controller optimizes the stimulation parameters, tailoring them to the patient's specific brain dynamics to ensure the most effective and personalized neurostimulation strategy.

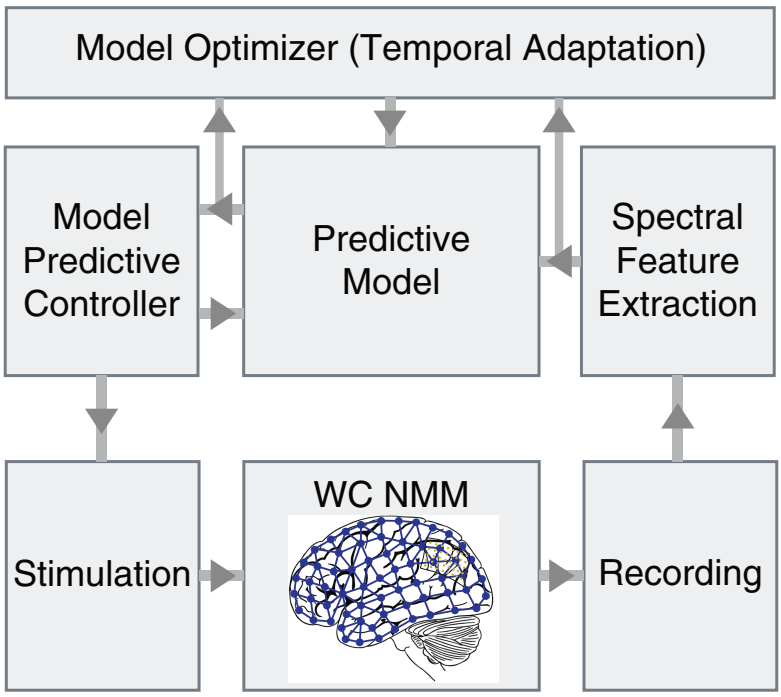

Figure 3-1 Simplified block diagram of the presented MPC-based adaptive neurostimulation framework.

To develop such a system, it is essential to have a highly accurate predictive model that can reliably represent seizure dynamics in patient's brain. However, seizure dynamics are a complex, networked, and nonlinear process that is still not fully understood [47], [113-114], and modeling it specifically for a patient is even more challenging.

3.1.1 Modeling System Dynamics

Several traditional system identification methods have been employed to model seizure dynamics using EEG data, including the auto-regressive moving-average model [115] and the fractional-order system model [116]. However, these methods have not yet been validated for their effectiveness in suppressing network-coupled seizure dynamics. Given the complexity of the brain and the significant variation in seizure types and symptoms, such simplifications—where these models reduce the intricate dynamics of brain activity to more manageable, but less accurate, representations—are unlikely to result in effective, patient-specific seizure control. With recent advancements in machine learning, data-driven models using neural networks have shown considerable promise in the field of system identification [117-119]. However, methods that are solely based on such models often suffer from a lack of interpretability, presenting challenges for safety-critical applications and making this an ongoing area of research.

Recently, the Koopman operator has gained attention as an effective method for capturing the intrinsic characteristics of nonlinear systems by enabling linear representation in an expanded observable space [120]. However, directly working with the Koopman operator in its original infinite-dimensional form often leads to representations that are computationally infeasible due to the high dimensionality. The Koopman operator acts on the space of all possible observables (functions of the system's state), making it inherently infinite-dimensional. To make the problem tractable, this operator must be approximated by a finite-dimensional matrix.

To address this, dimensionality reduction methods such as Dynamic Mode Decomposition (DMD) [121-122] and Extended Dynamic Mode Decomposition (EDMD) [123-125] are commonly used to approximate the Koopman operator with a finite-dimensional matrix. Unlike in DMD, where the observables, such as EEG recordings, are directly related to physical quantities and thus are physically meaningful, in EDMD, the observable functions are constructed using basis functions, such as Gaussian or polynomial functions. These basis functions do not necessarily correspond to physical quantities, which can lead to a more abstract representation of the system. However, the approximation performance can be sensitive to the choice of these basis functions, making their selection dependent on specialist expertise [126].

Driven by this challenge, recent research has focused on designing deep neural networks to automatically generate observable functions [127-128]. For instance, in [44], the authors developed a deep neural network to construct observables, demonstrating that their framework outperforms baseline models in data prediction when evaluated with the Koopman operator model. They tested their framework using both the Jansen-Rit model and the Epileptor model, which are well-suited for testing purposes. However, the parameters of these models were not optimized to generate patient-specific iEEG data. Their approach was based on a generic method of generating iEEG data, rather than utilizing a framework that produces iEEG features similar to those of a specific patient. Additionally, the efficiency of the stimulation waveform within their model was not discussed, leaving a critical aspect of closed-loop neurostimulation unexplored.

3.1.2 Overview of the Proposed Time-Adaptive Predictive Controller

In this chapter, we will present the design, development, and hardware implementation of an MPC-based closed-loop neurostimulation system and test it using the framework discussed in Chapter 2, which is optimized to generate synthetic iEEG data with features similar to those of a specific patient. The evaluation of different stimulations is conducted using the cost function outlined in Chapter 2.

Figure 3-2 shows a more detailed version of the proposed control system block diagram. At the core of this design is a predictive model responsible for forecasting the efficacy of stimulation waveforms on the NMM, which serves as a substitute for a specific patient. As shown, the ESD of the synthetic iEEG signal during a seizure episode is calculated using six FIR filters, and represent the current state of the NMM. Additionally, the control inputs are assumed as the amplitude and frequency used for generating the stimulation waveform. We only chose two parameters of the stimulation pulse (out of all parameters should in Figure 1-6 and Table 1-1) as a starting point, and the same approach could be taken with more parameters.

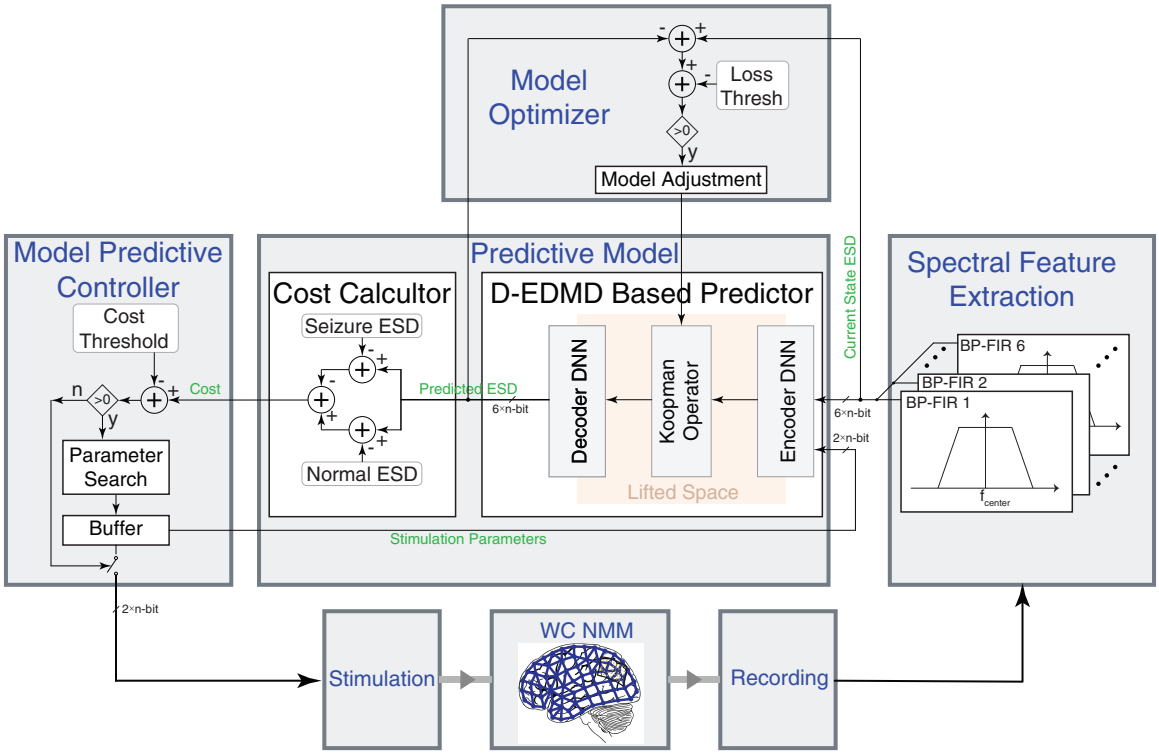

Figure 3-2 Detailed block diagram of the proposed stimulation control and adaptation process.

The current state and control inputs are fed into the Deep EDMD-based Koopman operator, which is tasked to predict the ESD of the signal in the next state. This block employs data-driven patient-specific encoders to perform prediction in a lifted linearized-dynamic space, and decoders to convert the prediction back to the input space.

The efficacy of the input (i.e., a set of stimulation parameters) is evaluated based on a cost value. This cost value, as will be described in detail, is calculated by comparing the predicted ESD with the ESD of the signal during the seizure and the ESD of the signal during a normal episode before seizure onset, both in six bands.

By leveraging the Koopman operator framework, Deep EDMD enables the representation of nonlinear seizure dynamics in a higher-dimensional linear space, providing a more interpretable model compared to deep neural networks (DNNs). Additionally, this approach helps reduce overfitting by integrating the system's dynamics into the model, ensuring better generalization, especially with limited data. Furthermore, Deep EDMD's ability to combine linear and nonlinear dynamics offers a structured approach to modeling complex systems, making it particularly advantageous over traditional DNNs.

Also shown in Figure 3.2, a Model Predictive Controller (MPC) is developed based on the predictions from the predictive model. This controller is designed to efficiently navigates the parameter space of stimulation waveforms, eliminating the need for extensive parameter sweeping or direct testing on patients.

Another important block shown in the figure is the optimizer, which enables the time-adaptivity of the predictive model. This block continuously evaluates the predictions of the predictive model based on the brain's response to stimulations. When the prediction error of the predictive model increases beyond a predefined threshold, which can occur due to factors such as resistance to certain stimulations, the optimizer retrain the predictive model. This block leverages our proposed deep EDMD approach by allowing the predictive model to switch between EDMD and Deep EDMD. In scenarios where updating the Koopman operators sufficiently improves prediction efficacy, the system can revert to EDMD without the need for further training to update observables. This capability significantly conserves power and time during real-time learning.

In this chapter, the implementation of predictive model and MPC in both software and hardware will be discussed.

3.2 Predictive model

We define $e = [\,\mathrm{ESD}_{0-5\mathrm{Hz}}\ \mathrm{ESD}_{5-10\mathrm{Hz}}\ \ldots\ \mathrm{ESD}_{25-30\mathrm{Hz}}]^{\mathrm{T}}$ to represent the current state vector, where $\mathrm{ESD}_{\mathrm{i-jHz}}$ denotes the ESD of the i-j Hz band, calculated based on one second of synthetic iEEG signal during seizure. The control inputs are defined as $u = [\mathrm{amplitude}\ \ \mathrm{frequency}]^{\mathrm{T}} \in U$. Seizure dynamics can be described as

$$\dot{e} = f_c(e, u), \tag{4-1}$$

in which ė represents changes in the current state, $e \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ are the state and the control input (for now, we set n=6 and m=2), respectively, and the mapping $f_c: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^s$ represents the dynamical function used to predict the system's response. In this work, We found s=20 to be high enough to achieve the desired predictive accuracy. With the predicted ESD of the signal, We calculate the predicted cost value associated with u using the cost function based on spectral bands described in Section 2.5.1.

We aim to find this dynamical function (i.e., the predictor) for the patient-specific NMM developed in chapter 2, without any prior knowledge of the internal parameters of the NMM and only through a data-driven approach.

3.2.1 EDMD-Based Koopman Operator for Nonlinear System Analysis

The Koopman operator is a powerful tool traditionally used to capture the intrinsic characteristics of unforced nonlinear dynamics by transforming them into a linear dynamical evolution [129-130]. With some modifications, the Koopman operator can also be extended to represent systems with control inputs. Consider the discrete-time system:

$$e_{n+1} = f(e_n, u_n) \tag{4-2}$$

Where n is the discrete-time index, e_n is the state vector at time step n, and u_n is the control input. According to the approach in [131], the Koopman operator can be generalized for the system in (4-2) by defining an extended state variable $z = [e^T\ u^T]^T$. The Koopman operator for this extended state z is then expressed as:

$$Kg(z_n) = g(z_{n+1}) \tag{4-3}$$

Here, K denotes the Koopman operator, which is inherently infinite-dimensional, and $g(z_n)$ represents an observable function in the lifted space. As mentioned previously, since working directly with an infinite-dimensional operator is impractical, and for practical purposes, particularly in controller design, it is necessary to approximate this infinite-dimensional Koopman operator with a finite-dimensional one. We use EDMD to achieve this approximation. We adapt $G(z_n) = [g_1(e_n)^T\ \dots\ g_L(e_n)^T\ u_n^T]^T \in \mathbb{R}^{L+m}$ as a group of observables for the practical calculation, where L is the number of observable functions on e, i.e., $g_1(e_n)$.

The main idea of computing the Koopman approximation with EDMD consists of two steps. Firstly, select the observable functions as basis functions, e.g. radial basis functions (RBF) with different kernel centers and widths. This is discussed in detail the next section. Secondly, compute a finite-dimensional approximation of the Koopman operator using the least-squares method. To this end, for forced dynamics of equation (4-1), we define [A B] as the first L rows of the approximated Koopman operator, where $A \in \mathbb{R}^{L\times L}$ and $B \in \mathbb{R}^{L\times m}$ are the system matrices of the lifted model.

$$G(e_{n+1}) = [A\ B]\begin{bmatrix} G(e_n) \\ u_n \end{bmatrix} \tag{4-4}$$

Matrices A and B can be computed via solving a least-squares problem based on a data set with M snapshots, i.e., $\{u_{[n]}, x_{[n]}, x^+\}^{M}_{n=1}$, where x^+ is the evolution of $x_{[n]}$ with $u_{[n]}$.

$$\min_{A,B} \sum_{t=1}^{M} \left\| G(e_{n+1}) - [A\ B]\begin{bmatrix} G(e_n) \\ u_n \end{bmatrix} \right\|^2 \tag{4-5}$$

3.2.2 Proposed Deep EDMD Framework for Koopman Operator Approximation

In this work, and considering the challenges described in previous sections, we propose a Deep EDMD to automatically construct an observable subspace of the Koopman operator. This approach allows us to approximate the system dynamics using deep learning techniques. The dynamics approximated by Deep EDMD can be represented as follows:

$$\varphi_{e,n+1} = K\Psi(e_n, \theta_e, u_n) \tag{4-6}$$

$$\hat{e}_n = \tilde{\Psi}(\varphi_{e,t}, \theta_d) \tag{4-7}$$

Here's a breakdown of the notation:

- $K = [A\ B] \in \mathbb{R}^{L\times N}$, where $N = L + m$. The matrix K is the Koopman operator in its finite-dimensional approximation.

- $\varphi_{e,n} = \varphi_e(e_n, u_n) \in \mathbb{R}^L$ represents the encoded state at time step n. The function $\varphi_e(\cdot)$ is the encoder, which is parameterized by weights and biases θ_e.
- $\Psi(e_n, \theta_e, u_n) = [\varphi_e(e_n, u_n)^T \, u_n^T]^T \in \mathbb{R}^N$, is the augmented observable that combines the encoded state and the control input u_n.
- $\widetilde{\Psi}(\varphi_{e,t}, \theta_d)$ represents the decoder, parameterized by weights parameterized with weights and biases θ_d, which reconstructs the estimated state $\hat{e}_n$ from the encoded state $\varphi_{e,t}$.

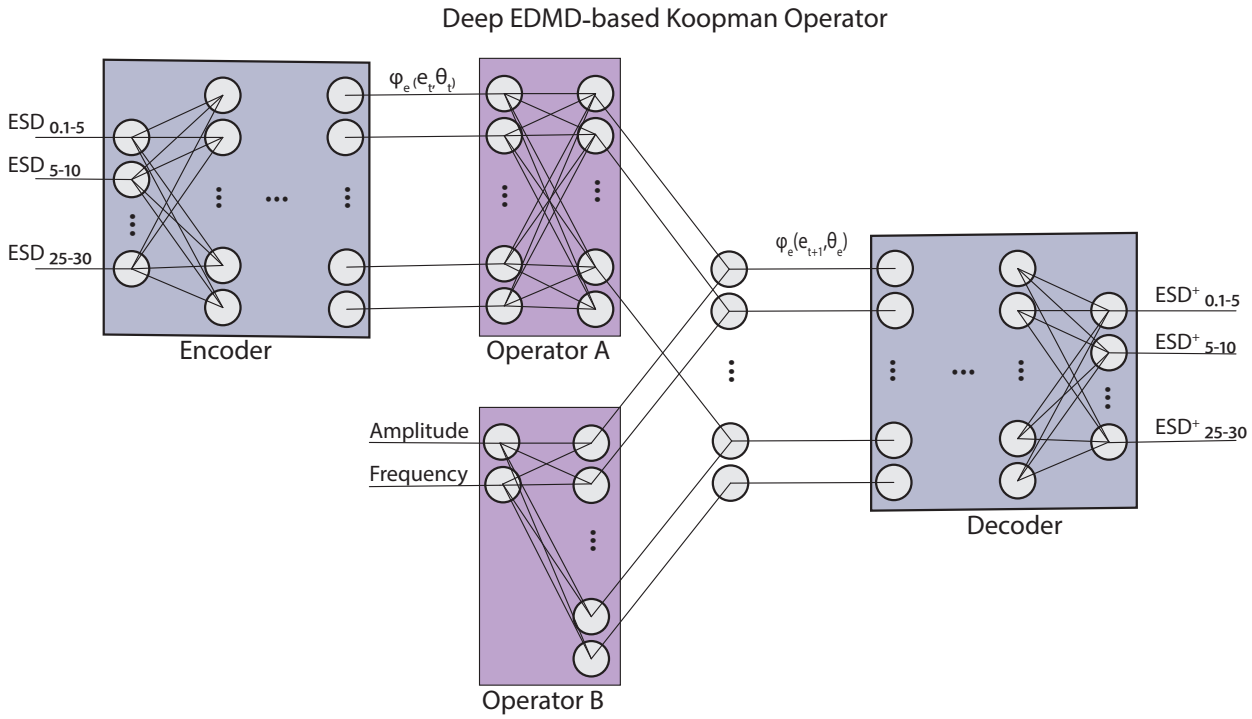

Figure 3-3 Simplified block diagram of the DNN-based Encoder and Decoder, as well as the Koopman operator used for the predictive model implementation.

As shown in Figure 3-3, the Deep EDMD algorithm relies on an autoencoder structure. The encoder, i.e., φ_e, including a neural network constructed by fully-connected layers, is in charge of mapping the original state to a lifted observable space. The weights A and B are

placed connecting to the last layer of the encoder without activation functions. Similar to the encoder, the decoder consists of a fully-connected neural network, responsible recovering the original state from the lifted observable space. To be specific, at any hidden layer L, the output can be described as

$$\bar{y}_*^{(L)} = \sigma_*^{(L)}(W_*^{(L)} y_*^{(L-1)} + b_*^{(L)}) \tag{4-8}$$

where $* = e, d$ in turn stands for the subscripts for the encoder or decoder, $W_*^{(L)} \in \mathbb{R}^{n_L \times n_{L-1}}$ and $b_*^{(L)} \in \mathbb{R}^{n_L}$ are the weight and bias of the hidden layer L, where n_L represents the number of neurons of the hidden layer L. $\sigma^{(L)}$ denotes the activation function of the hidden layer L, L = [1:H], where H is the number of hidden layers of the encoder and decoder. In the encoder,$\bar{y}_e^{(0)}$, $\sigma_e^{(l)}$, L = [1:H], is designed using a rectified linear unit (ReLU) [132]. As for the decoder, $\sigma_d^{(L)}$, L = [1:H] uses ReLU as the activation function. The lifted state can be obtained by the encoder, i.e.,

$$\varphi_e(e_n, \theta_e) = [e_n^T \left(\bar{y}_e^{(H)}\right)^T]^T \tag{4-9}$$

Where $\bar{y}_e^{(H)} \in \mathbb{R}^{L-n}$ is the output of the last layer of the neural network in the encoder, which can be computed with (4-8) based on e_n. Similarly, the reconstructed state can be computed by the decoder based on the lifted state (4-7).

For training the deep EDMD model, we defined the loss function as the sum of the errors in ESDs prediction:

$$Loss = \|e_{n+1} - \hat{e}_{n+1}\| \tag{4-10}$$

Here $\hat{e}_{n+1}$ is the predicted ESD vector after the stimulation is applied, e_{n+1} are the true data that we calculated with NMM. Adam solver using gradient descent is adopted in this project to minimize the loss function and train encoder and decoder [133].

3.2.3 Parameter Setting and Training of Deep EDMD

In the Deep EDMD model, the encoder was designed with five layers, following the structure [6, L, L, L, L], while the decoder was structured as [L, L, L, L, 6]. In our simulations, the value of L, representing the number of neurons per layer, was set to 20. The model was trained with a batch size of 200, meaning it processed 200 samples at a time before updating its parameters. Training occurred over 400 epochs, with the model going through the entire dataset 400 times to refine its performance. A small learning rate of 10^{-6} was used to make gradual adjustments, ensuring precise learning. All the weights were initialized with a uniform distribution limiting each weight in the range $[-\omega\ \omega]$ for $\omega = \frac{1}{\sqrt{\alpha}}$, where α is the number of the network layer [134].

Since Deep EDMD utilizes neural networks, it was trained using Python API in TensorFlow [135] framework with an Adam solver [136], and using an NVIDIA GeForce GTX 2080 Ti GPU.

3.3 Performance Evaluation of Predictive Model

3.3.1 Data Collection and Preprocessing

We validated our Predictive Model using the proposed Patient-Specific WC-NMM described in Chapter 2. As the first step of testing, we chose one of the patients (patient #10) and the patient-specific optimized NMM was fine-tuned to generate iEEG signals with low error in the frequency domain when compared to pre-recorded iEEG from this patient in the SWEC-ETHZ dataset. There are 1,245 seconds of seizure time for this patient in the dataset, spanning 17 seizures with an average duration of 72 seconds each. An important objective in the design of the presented framework is to ensure that this amount of pre-recorded iEEG data is sufficient for fully training the model.

We create a dataset with our NMM consists of 88 episodes, each containing a 15-second iEEG recording divided into three segments: the first 1 second represents a normal episode, the next 1 second represents a seizure episode without stimulation, and the final 13 seconds

represent a seizure episode with 13 random wave stimulations. The random wave stimulation had an amplitude range of 0 to 5 mA and a frequency range of 0 to 200 Hz.

In these 88 episodes, oscillations at different frequencies were generated during the seizure episodes to mimic the various seizure oscillations observed in Patient 10 (discussed in Chapter 2). We calculated the energy in six frequency bands (0-5 Hz, 5-10 Hz, 10-15 Hz, 15-20 Hz, 20-25 Hz, 25-30 Hz) for every second of the 15-second recording using FIR filters, as described in previous chapter.

To create the dataset, we followed this process:

- N_data (13×6 matrix) stores the 6 ESD values calculated from the first second, repeated 13 times.
- S_data (13×6 matrix) stores the 6 energy bands calculated from the second second, repeated 13 times.
- St_data (13×6 matrix) stores the ESD values calculated from the remaining 13 seconds.
- C_data (13×2 matrix) stores the stimulation parameters corresponding to each second of the 13-second period.

This process was repeated 88 times, resulting in a dataset with 1144 data points. To improve the quality of the St_data for training, normalization was applied as a preprocessing

step. The data was adjusted to a consistent range between 0 and 1, making it more suitable for model training. The dataset was then divided into 50% for training and 50% for validation. We used separate data for training and validation, ensuring that the model's performance is evaluated on unseen data during the training process. Testing was conducted on a larger dataset generated by the NMM, where the amplitude and frequency of stimulation waveforms were swept and applied to three different seizure-like oscillatory behaviors, ensuring thorough evaluation of all parameters.

3.3.2 Training and Testing Process for Prediction Accuracy Evaluation

Figure 3-4 illustrates the Training and Validation Losses over 400 epochs, with the loss defined in Equation 4-10, plotted on a logarithmic scale to capture the wide range of loss values. Initially, both losses decrease rapidly, indicating that the model is quickly learning from the data. As training progresses, the losses converge, with the final loss reaching below 10^{-4}, which is low compared to the range of the input values (0-1). This low loss indicates that the model has effectively minimized the error and is performing well. The close alignment between training and validation losses throughout the process suggests strong generalization without overfitting, confirming that the chosen model architecture and training strategy are highly effective.

Figure 3-5 compares the one example of the observed (i.e., generated by the NMM) ESD of the post-stimulation signal during a seizure with the predicted denormalized ESDs generated by the Deep EDMD-based Koopman Operator, when given the same stimulation parameters.

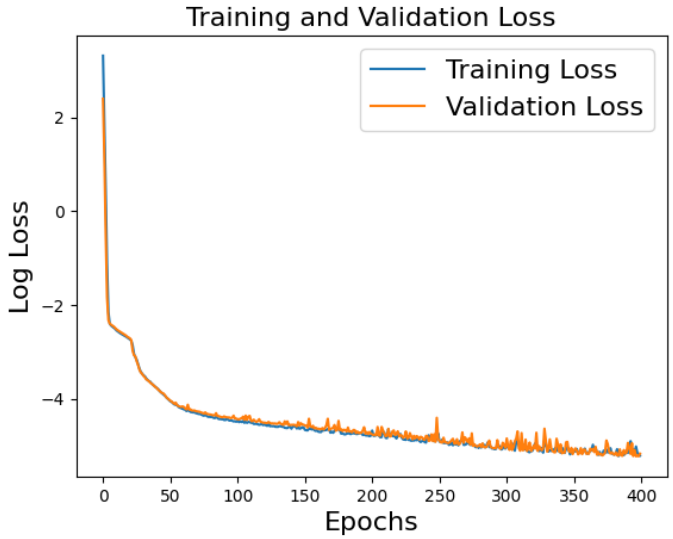

Figure 3-4 Training and Validation Losses over 400 epochs for the developed predictive model.

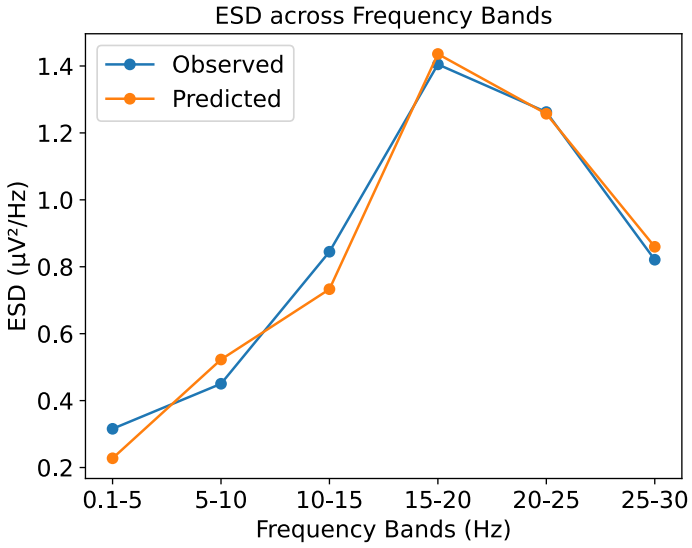

Figure 3-5 An example comparison of observed and predicted denormalized ESDs during a seizure in response to specific stimulation.

3.3.3 Stimulation Efficacy Prediction Performance

To assess the accuracy of our predictive model in forecasting stimulation efficacy and identifying optimal stimulation parameters, we created a large dataset using the NMM. We performed a parameter sweep across the parameter space, sampling 100 data points in the frequency range of 0-200 Hz and 100 data points in the amplitude range of 0-5 mA, while our NMM exhibited specific seizure-like oscillations (with cell types having 15 ms time constants generating seizure-like activity). For each combination of frequency and amplitude, we calculated the corresponding cost values.

The cost function, as detailed in Chapter 2, assigns each frequency band a weight proportional to the magnitude of its energy increase during seizure activity. We then compute the difference in the ESD of the signal between the stimulated state and the normal state for each frequency band, multiplying this difference by the corresponding band weight. Finally, we sum the absolute values across all bands to produce a single cost value.

Due to the presence of noise, the optimal set of parameters varied in each experiment. To capture this variability, we repeated the process five times, observing the variation of parameters at the optimal points. This process was further repeated with two other different seizure-like oscillations (cell types with 10 ms and 20 ms time constants generating oscillatory behaviors).

To test the performance of our model, the predicted cost value associated with each set of stimulation parameters in the dataset is calculated using the output of the Deep EDMD-based Koopman Operator, which predicts the ESD of the signal when stimulation corresponding to these parameters is applied. Since the model was trained with a normalized dataset, the output of the model is denormalized because the cost function compares the difference in ESDs with normal episodes, requiring real values for accurate cost value prediction. Figure 3-6(a) presents heatmaps of the predicted cost values generated by our predictive model, while Figure 3-6(b) displays the cost values obtained by applying stimulations directly to the NMM. This process was repeated with two other different seizure-like oscillations (cell types with 10 ms and 20 ms constants generating oscillatory behaviors). Table 3-1 shows the results of the optimal stimulation parameters and the predicted optimal values by Deep EDMD-based Koopman Operator for each seizure.

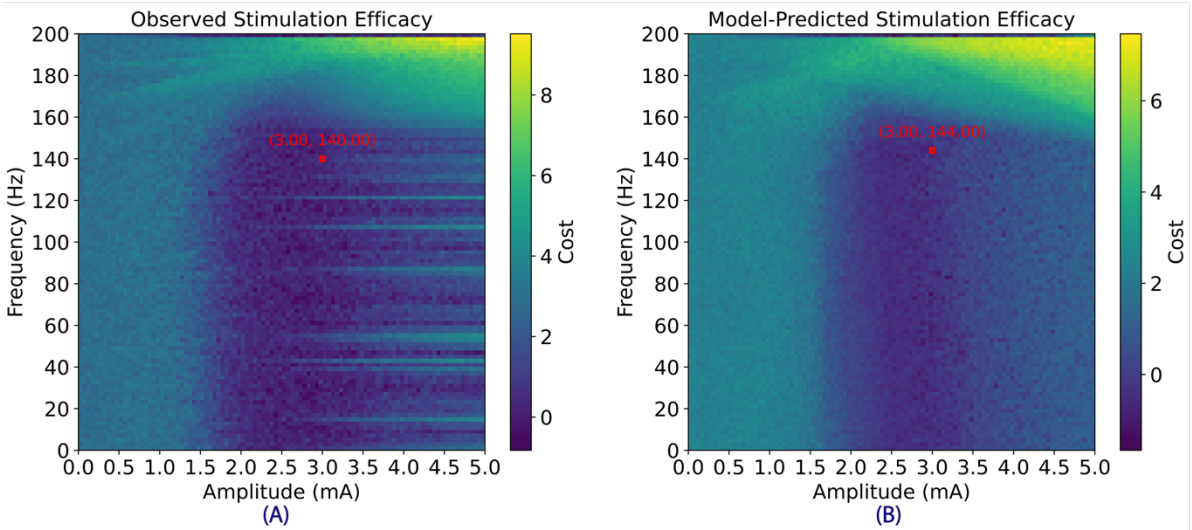

Figure 3-6 (a) Heatmap of predicted cost values by the predictive model, and (b) Heatmap of cost values calculated from direct stimulation of the NMM.

Table 3-1 Results of Observed and Predicted Optimal Stimulation Parameters for Each Seizure.

Time constant of Cell Type in Seizure Mode	Observed Optimal Stimulation Parameters (Amplitude, Frequency)	Predicted Optimal Stimulation Parameters (Amplitude, Frequency)
20 ms	(2.95±0.05 mA, 60±12 Hz)	(2.95 mA, 66 Hz)
15 ms	(3.00±0.05 mA, 144±12 Hz)	(3.00 mA, 144 Hz)
10 ms	(3.55±0.1 mA, 104±18 Hz)	(3.75 mA, 126 Hz)

As shown in Table 3-1, the predictive model was successful in accurately identifying the optimal stimulation parameters for 2 out of 3 seizure oscillations in this patient. The predicted values closely matched the observed values for the 20 ms and 15 ms time constants, while there was a slight discrepancy in the seizure created with a 10ms-time-constant cell type.

3.4 MPC with Deep EDMD-Koopman based Operator for Closed-Loop Neurostimulation

As depicted in Figure 3-2, the MPC is tasked to optimize the stimulation parameters based on the predictive model's estimation of stimulation efficacy. The key benefit is that these evaluations occur in silico, without the need to actually apply the stimulation parameters to the brain. This approach not only accelerates the process of identifying an optimal solution but also enhances safety by avoiding unnecessary stimulation trials. Furthermore, the reliance on the predictive model eliminates concerns associated with simultaneous recording and stimulation, such as the need for high dynamic range and high linearity in recording circuits,

and the challenge of removing stimulation artifacts. Although the chosen parameters are ultimately tested in reality, and the model is adjusted accordingly, the MPC-based method significantly reduces the number of iterations required to reach an optimal point. Compared to traditional iterative approaches guided by algorithms like gradient descent, the MPC method is more efficient, requires less power, and offers a safer alternative for patient-specific optimization.

The optimization process begins by predicting the cost value of an initial set of parameters (i.e., the initial point) using the proposed predictive model. Subsequently, this procedure is repeated for 10 randomly selected points in the vicinity of the initial point, updating the initial point to the one that produces the lowest cost value. This iterative process continues to seek the lowest cost, fine-tuning the stimulation parameters. In each iteration, if none of the 10 points yields a lower cost value compared to the previous iteration, the range for generating the 10 points is expanded, and the process repeats. The process concludes when the cost function falls below a predefined threshold value, determined based on the average cost function value during non-seizure episodes.

3.4.1 MPC Performance Evaluation

To validate the proposed MPC, we tested it to find the optimal stimulation parameters for three different seizure-like activities in the NMM. We compared the final stimulation

parameters found by the MPC with the data points that had the lowest cost values determined after sweeping the entire parameter space. The MPC successfully identified optimal stimulation with a difference less than the sweeping step size. Figure 3-7(a) illustrates three example path trajectories that the optimizer followed before converging to an optimal set of parameters. To clarity, only the top-performing set of parameters among the 10 sets evaluated at each step is displayed. It demonstrates that in these three processes, the MPC successfully identified the predicted optimal stimulation parameters (as listed in Table 3-1) with a difference smaller than the sweeping step size in fewer than 20 steps, reaching the lowest possible cost value.

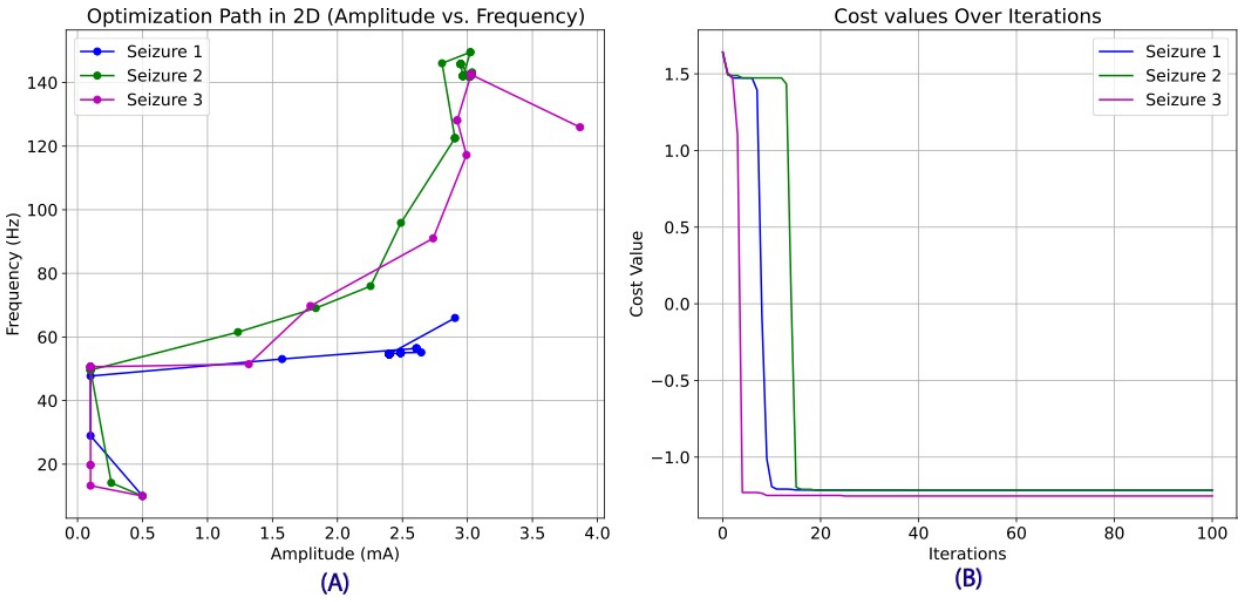

Figure 3-7 (a) Three examples of the trajectory of stimulation pulse parameter optimization steps from the start to the optimal point. (b) Calculated optimization cost in each step of the process for 3 different seizures.

3.5 Hardware Implementation

The Predictive Model and Model Predictive Control (MPC) block were implemented for online testing on a Xilinx Artix UltraScale+ FPGA. The weights and biases for the encoder and decoder were calculated through offline training for each patient, and then were loaded onto the FPGA. This setup enables the Predictive Model to handle the real-time execution of the predictive model and MPC once the optimizer is implemented, ensuring efficient and timely processing of iEEG signals for seizure suppression.

For numerical operations in the model, a custom package was defined to handle fixed-point arithmetic, using a 12-bit signed format with 6 fractional bits. The choice of this specific bit configuration was determined based on the minimum and maximum numbers involved in our implementation, ensuring that the fixed-point arithmetic can accurately represent the necessary range of values while maintaining precision and efficiency in the hardware implementation on the FPGA.

Figure 3-7 shows the block diagram of the hardware implementation on the FPGA. The inputs are the seizure/non-seizure (S/N) label provided by the detection block and the serialized iEEG signal. The main blocks in the diagram include the Deserializer, Memory, FIR Band-Pass Filters (FIR BPF1), Encoder, Koopman Operators, Decoder, Cost Function, Model Predictive Control (MPC) block and Serializer.

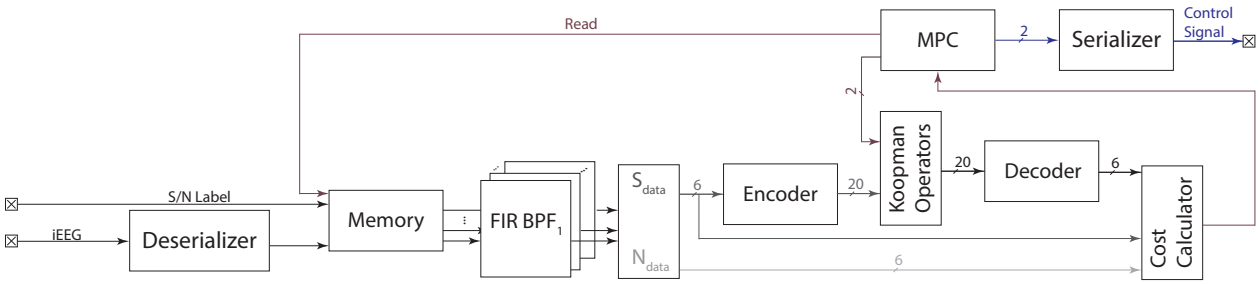

Figure 3-8 Block diagram of the hardware implementation for the predictive model and MPC on the Xilinx Artix UltraScale+ FPGA.

The Memory block in the system is designed to handle and store the iEEG signal with two primary functions. It operates with two clocks: a sampling clock for continuously receiving and updating the iEEG signal, and an internal clock for sending the stored data to the next stage of processing. The Memory block maintains the most recent two seconds of the iEEG signal, updating this data in a rolling manner—each new sample input causes the oldest sample to be removed. When the S/N label is triggered, indicating the detection of a seizure, the Memory block promptly sends the stored two-second signal to the FIR filter blocks for further analysis. This setup ensures that relevant iEEG data is always available and ready for processing immediately upon seizure detection.

Six FIR band-pass filters were implemented, each with 50 taps, to calculate the Energy Spectral Density (ESD) of each second of the two-second input signal. The choice of the number of filters and the number of taps is discussed in detail in Chapter 2. After the FIR band-pass filters calculate the ESD for each second of the two-second input signal, the ESD values are sent to a multiplexer. The multiplexer divides the ESD of the first second, identifying it as the normal label (N_data), while the ESD of the second second, associated with the seizure

label, is designated as S_data. This separation allows for further analysis and processing of both normal and seizure-related data in the subsequent stages of the system.

The S_data is then sent to the Encoder, which consists of four layers of a fully connected neural network. Each layer is implemented using a matrix multiplier, with summation biases applied to transform the input data into a latent space. The encoded data is then sent to the Koopman Operators block, along with a control signal from the MPC, to transform the data to the next state in the latent space. The Koopman Operators block is also implemented using a matrix multiplier. The next state in the latent space is subsequently sent to the Decoder. The decoded data represents the predicted ESD of the signal when the stimulation , corresponding to the amplitude and frequency specified in the control signal, is applied.

The predicted ESD, along with N_{data} and S_{data}, are inputs to the Cost Function block to calculate the predicted corresponding control signal as defined by the MPC. The resulting cost value is then sent to the MPC, which uses this information to update the control signal and search for the most optimal stimulation parameters within the parameter space. Once the optimal parameters are identified, they are sent to the stimulator unit through the serializer block. Simultaneously, the "read" signal is triggered, allowing new signals to be retrieved from the Memory block, enabling the continuous optimization of the stimulation parameters. This loop ensures that the system consistently adapts to the ongoing iEEG signals, maintaining effective seizure suppression.

3.5.1 Synthesis Report

The adaptive controller VHDL code was synthesized using Vivado, and subsequently, a power estimation was performed. Power consumption reported by Vivado was <1mW for total dynamic power, with no internal or leakage power recorded. Since the Vivado tool was incapable of reporting sub-mW power estimates a more accurate estimation was required. Therefore, the design was synthesized using Synopsys with a standard CMOS 130nm technology.

The detailed results of the synthesis and power estimation, including cell usage and key parameters, are presented in Tables 3-2 and 3-3 below.

Table 3-2 Cell Usage Summary

Cell Type	Count
BUFG	2
CARRY8	35
LUT1	27
LUT2	82
LUT3	20
LUT4	90
LUT5	56
LUT6	148
FDCE	3
LD	126
IBUF	3
OBUF	25

Table 3-3 Key Parameters of the Adaptive Controller

Parameter	Value
Input Clock Frequency	1024 Hz (Sampling Rate of iEEG)
Internal Clock Frequency	1 MHz
Delay for Finding Optimal Stimulation	1200 μs after the first second of the seizure

Table 3-4 Area of Different Hardware Blocks in the Adaptive Controller System.

Block	Area (μm^2)
Memory	801.172
FIR filters	54350.124
Multiplexer	623.225
Encoder	658.591
Koopman	6394.105
Decoder	658.591
Cost Calculator	47206.392
MPC	6231.155
Total	116,923.36

Table 3-5 Power Consumption of Different Hardware Blocks in the Adaptive Controller System.

Block	Power Consumption (μW)
Memory	0.5076
FIR filters	31.324
Multiplexer	0.321
Encoder	0.54442
Koopman	9.0651
Decoder	0.54442
Cost Calculator	24.927
MPC	29.854
Total	97.09

It is important to note that no specific power optimization efforts were applied to the adaptive controller design.

For each electrode, this adaptive controller can optimize the stimulation parameters. This approach can be expanded to multi-electrode stimulation, where an additional block is required to evaluate the cost values across electrodes and select the optimal electrode for stimulation.

Chapter 4 Conclusions and Future work

4.1 Summary and Conclusions

This thesis presented a framework for the development and optimization of closed-loop neurostimulators tailored to conduct time-adaptive, patient-specific stimulation.

In Chapter 2, we tackled two challenges mentioned in the introduction for this purpose. The first challenge addressed was the lack of comprehensive datasets necessary for data-driven methods, such as machine learning algorithms. The second challenge was the absence of robust indicators for the immediate evaluation of stimulation efficacy. To this end, we developed a patient-specific NMM that can be fine-tuned to generate iEEG signals with high spectral correlation to real pre-recorded iEEG signals from individual patients. This model was meticulously designed to accurately model the various oscillatory behaviors that occur during a patient's seizure and to simulate the neuronal response to different electrical stimulation parameters. By fine-tuning the NMM for each patient, we ensured that the model

could effectively model the unique dynamics of seizure activity, thereby providing a powerful tool for developing patient-adaptive neurostimulation therapies.

Chapter 2 also explored the existence of optimal stimulation parameters and introduced two indicators to evaluate the efficacy of these stimulations. We thoroughly discussed the reliability of both indicators, identifying one as a more reliable and comprehensive measure of the iEEG signal's spectral characteristics under stimulation. The practical implementation of this indicator in hardware was also addressed, emphasizing its relevance for real-time applications.

In Chapter 3, we tackled two other critical challenges in developing and optimizing closed-loop neurostimulators tailored to conduct time-adaptive, patient-specific stimulation: the large parameter space of stimulation and the non-determinism in seizure control. To address these challenges, we proposed an MPC-based closed-loop neurostimulation system.

For the predictive model, we introduced the Koopman-deep EDMD approach, which aims to identify the dynamical functions underlying seizure activity. This model is capable of adapting to changes in the brain's dynamics, which can occur due to various factors such as aging. Based on the predictions from this model, a MPC was developed. This controller efficiently navigates the parameter space of stimulation waveforms, eliminating the need for extensive parameter sweeping or direct testing on patients.

Both the predictive model and the MPC were implemented on an FPGA as well as synthesized using a standard CMOS 130nm process. A sub-mW power consumption in both approaches was observed, which aligns well with our power budget constraints.

Future Work

In the immediate future, expanding the patient-specific NMM into a larger network is a key area of focus. This expansion aims to model seizure propagation across different brain regions, thereby improving our understanding of seizure dynamics and enhancing prediction and control strategies. Additionally, implementing the optimizer block in hardware, which enable online learning to automatically adjust stimulation parameters in response to temporal changes is expected to significantly enhance the utility of the adaptive controller.

Moreover, there is a need to optimize additional stimulation parameters, such as waveform shape, pulse duration, and intensity. This will allow for the development of more comprehensive and personalized neurostimulation therapies.

Looking toward the long-term future, extending the predictive model to support multi-step predictions is a critical goal. This would enable the MPC to plan and execute a sequence of control actions over an extended period, thereby improving the system's ability to manage complex and evolving seizure dynamics. Another long-term objective is the development of a multi-channel MPC capable of analyzing iEEG signals from multiple brain regions

simultaneously. This approach could identify the optimal location for applying stimulation, potentially enhancing the efficacy of treatment. Additionally, the ultimate goal is to integrate all critical components—including the recording block, seizure detection, adaptive controller, and stimulator—into a single, cohesive system for real-time, closed-loop operation. Finally, conducting in vivo testing of the fully integrated system is crucial for validating its effectiveness. Such testing will provide key insights into the system's performance in real-world clinical settings, paving the way for future clinical applications.

References

[1] C. J. Murray and A. D. Lopez, "The global burden of disease: a comprehensive assessment of mortality and disability from diseases, injuries, and risk factors in 1990," Organization, WH, 1996.

[2] R. Kale, "Bringing epilepsy out of the shadows: wide treatment gap needs to be reduced," BMJ, vol. 315, no. 7099, pp. 2-3, 1997.

[3] P. Jiruska, M. De Curtis, J. G. Jefferys, C. A. Schevon, S. J. Schiff, and K. Schindler, "Synchronization and desynchronization in epilepsy: controversies and hypotheses," The Journal of Physiology, vol. 591, no. 4, pp. 787-797, 2013.

[4] E. H. Reynolds, "Introduction: epilepsy in the world," Epilepsia (Series 4), vol. 43, 2002.

[5] R. S. Fisher et al., "Instruction manual for the ILAE 2017 operational classification of seizure types," Epilepsia, vol. 58, no. 4, pp. 531-542, 2017.

[6] R. S. Fisher et al., "Operational classification of seizure types by the International League Against Epilepsy: position paper of the ILAE Commission for Classification and Terminology," Zeitschrift für Epileptologie, vol. 31, pp. 272-281, 2018.

[7] V. Salanova, "Deep brain stimulation for epilepsy," Epilepsy & Behavior, vol. 88, pp. 21-24, 2018.

[8] N. Klinger and S. Mittal, "Deep brain stimulation for seizure control in drug-resistant epilepsy," Neurosurgical Focus, vol. 45, no. 2, pp. E4, 2018.

[9] D. San-Juan et al., "Neuromodulation techniques for status epilepticus: A review," Brain Stimulation, vol. 12, no. 4, pp. 835-844, 2019.

[10] C.-C. Chiang et al., "Seizure suppression by high frequency optogenetic stimulation using in vitro and in vivo animal models of epilepsy," Brain Stimulation, vol. 7, no. 6, pp. 890-899, 2014.

[11] S. A. Desai et al., "Asynchronous distributed multielectrode microstimulation reduces seizures in the dorsal tetanus toxin model of temporal lobe epilepsy," Brain Stimulation, vol. 9, no. 1, pp. 86-100, 2016.

[12] H. Kassiri, A. Muneeb, R. Salahi, and A. Dabbaghian, “Closed-Loop Implantable Neurostimulators for Individualized Treatment of Intractable Epilepsy: A Review of Recent Developments, Ongoing Challenges, and Future Opportunities,”, Accepted (Early Access), IEEE Transactions on Biomedical Circuits and Systems

[13] A. Tendler et al., "Deep transcranial magnetic stimulation (dTMS)–beyond depression," Expert Review of Medical Devices, vol. 13, no. 10, pp. 987-1000, 2016.

[14] D. San-Juan et al., "Transcranial direct current stimulation in epilepsy," Brain Stimulation, vol. 8, no. 3, pp. 455-464, 2015.

[15] E. Ben-Menachem et al., "Vagus nerve stimulation for treatment of partial seizures: 1. A controlled study of effect on seizures," Epilepsia, vol. 35, no. 3, pp. 616-626, 1994.

[16] R. S. Fisher et al., "Automatic vagus nerve stimulation triggered by ictal tachycardia: clinical outcomes and device performance—the US E-37 trial," Neuromodulation: Technology at the Neural Interface, vol. 19, no. 2, pp. 188-195, 2016.

[17] M. J. Morrell, "Responsive cortical stimulation for the treatment of medically intractable partial epilepsy," Neurology, vol. 77, no. 13, pp. 1295-1304, 2011.

[18] R. Fisher et al., "Electrical stimulation of the anterior nucleus of thalamus for treatment of refractory epilepsy," Epilepsia, vol. 51, no. 5, pp. 899-908, 2010.

[19] T. L. Skarpaas and M. J. Morrell, "Intracranial stimulation therapy for epilepsy," Neurotherapeutics, vol. 6, pp. 238-243, 2009.

[20] M. T. Salam et al., "Rapid brief feedback intracerebral stimulation based on real-time desynchronization detection preceding seizures stops the generation of convulsive paroxysms," Epilepsia, vol. 56, no. 8, pp. 1227-1238, 2015.

[21] M. T. Salam, J. L. Perez Velazquez, and R. Genov, "Seizure suppression efficacy of closed-loop versus open-loop deep brain stimulation in a rodent model of epilepsy," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 24, no. 6, pp. 710-719, 2015.

[22] A. Hartshorn and B. Jobst, "Responsive brain stimulation in epilepsy," Therapeutic Advances in Chronic Disease, vol. 9, no. 7, pp. 135-142, 2018.

[23] M. Parastarfeizabadi and A. Z. Kouzani, "Advances in closed-loop deep brain stimulation devices," Journal of Neuroengineering and Rehabilitation, vol. 14, pp. 1-20, 2017.

[24] National Institute of Health, "Medtronic," available at: https://commonfund.nih.gov/sites/default/files/Medtronic. [Accessed: Sep. 02, 2024].

[25] D. R. Nair et al., "Nine-year prospective efficacy and safety of brain-responsive neurostimulation for focal epilepsy," Neurology, vol. 95, no. 9, pp. e1244-e1256, 2020.

[26] H. D. Simpson et al., "Practical considerations in epilepsy neurostimulation," Epilepsia, vol. 63, no. 10, pp. 2445-2460, Oct. 2022.

[27] S. K. Arfin and R. Sarpeshkar, "An energy-efficient, adiabatic electrode stimulator with inductive energy recycling and feedback current regulation," IEEE Transactions on Biomedical Circuits and Systems, vol. 6, no. 1, pp. 1-4, Oct. 2011.

[28] H. Kassiri et al., "Arbitrary-waveform electro-optical intracranial neurostimulator with load-adaptive high-voltage compliance," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 27, no. 4, pp. 582-593, Feb. 2019.

[29] E. Noorsal et al., "A neural stimulator frontend with high-voltage compliance and programmable pulse shape for epiretinal implants," IEEE Journal of Solid-State Circuits, vol. 47, no. 1, pp. 244-256, Sep. 2011.

[30] M. Azin et al., "A battery-powered activity-dependent intracortical microstimulation IC for brain-machine-brain interface," IEEE Journal of Solid-State Circuits, vol. 46, no. 4, pp. 731-745, Mar. 2011.

[31] M. Azin and P. Mohseni, "A high-output-impedance current microstimulator for anatomical rewiring of cortical circuitry," in 2008 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2502-2505, May 2008.

[32] M. Ghovanloo and K. Najafi, "A wireless implantable multichannel microstimulating system-on-a-chip with modular architecture," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 15, no. 3, pp. 449-457, 2007.

[33] H. M. Lee et al., "A power-efficient switched-capacitor stimulating system for electrical/optical deep brain stimulation," IEEE Journal of Solid-State Circuits, vol. 50, no. 1, pp. 360-374, Jan. 2015.

[34] F. Eshaghi et al., "A 24-channel neurostimulator IC with one-shot impedance-adaptive channel-specific charge balancing," in 2021 IEEE Custom Integrated Circuits Conference (CICC), 2021.

[35] F. Eshaghi et al., "A neurostimulator IC with impedance-aware dynamic-precision one-shot charge balancing," IEEE Solid-State Circuits Letters, vol. 4, pp. 202-205, 2021.

[36] R. Ranjandish and A. Schmid, "An active charge balancing method based on self-oscillation of the anodic current," in 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2016.

[37] F. Eshaghi, E. Najafiaghdam, and H. Kassiri, "A 24-channel neurostimulator IC with channel-specific energy-efficient hybrid preventive-detective dynamic-precision charge balancing," IEEE Access, vol. 9, pp. 95884-95895, 2021.

[38] D. Kumsa et al., "Electrical neurostimulation with imbalanced waveform mitigates dissolution of platinum electrodes," Journal of Neural Engineering, vol. 13, no. 5, pp. 054001, 2016.

[39] H. Kassiri et al., "An impedance-tracking battery-less arbitrary-waveform neurostimulator with load-adaptive 20V voltage compliance," in ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference, 2016.

[40] H. Kassiri et al., "Inductively-powered direct-coupled 64-channel chopper-stabilized epilepsy-responsive neurostimulator with digital offset cancellation and tri-band radio," in ESSCIRC 2014 - 40th European Solid-State Circuits Conference (ESSCIRC), 2014.

[41] Y. Lin and Y. Wang, "Neurostimulation as a promising epilepsy therapy," Epilepsia Open, vol. 2, no. 4, pp. 371-387, 2017.

[42] Z. Liang et al., "Online learning Koopman operator for closed-loop electrical neurostimulation in epilepsy," IEEE Journal of Biomedical and Health Informatics, vol. 27, no. 1, pp. 492-503, 2022.

[43] M. Beudel and P. Brown, "Adaptive deep brain stimulation in Parkinson's disease," Parkinsonism & Related Disorders, vol. 22, pp. S123-S126, 2016.

[44] R. Salahi and H. Kassiri, "NMM-Based Patient-Specific Temporally-Adaptive Stimulation Optimization for Seizure Control," 2024 IEEE International Symposium on Circuits and Systems (ISCAS), Singapore, Singapore, 2024, pp. 1-5.

[45] V. K. Jirsa et al., "On the nature of seizure dynamics," Brain, vol. 137, no. 8, pp. 2210-2230, 2014.

[46] C. Diaz Verdugo et al., "Glia-neuron interactions underlie state transitions to generalized seizures," Nature Communications, vol. 10, no. 1, pp. 3830, 2019.

[47] M. Arrais et al., "Design of optimal multi-site brain stimulation protocols via neuro-inspired epilepsy models for abatement of interictal discharges," Journal of Neural Engineering, vol. 18, no. 1, pp. 016024, 2021.

[48] T. Moeinfard, G. Zoidl, and H. Kassiri, "A SAR-Assisted DC-Coupled Chopper-Stabilized 20μs-Artifact-Recovery ΔΣ ADC for Simultaneous Neural Recording and Stimulation," in 2022 IEEE Custom Integrated Circuits Conference (CICC), 2022.

[49] M. R. Pazhouhandeh et al., "Opamp-less sub-μW/channel Δ-modulated neural-ADC with super-GΩ input impedance," IEEE Journal of Solid-State Circuits, vol. 56, no. 5, pp. 1565-1575, 2020.

[50] M. R. Pazhouhandeh et al., "Artifact-tolerant opamp-less delta-modulated bidirectional neuro-interface," in 2018 IEEE Symposium on VLSI Circuits, 2018.

[51] T. Moeinfard and H. Kassiri, "A 200GΩ-Z IN, <0.2%-THD CT-ΔΣ-Based ADC-Direct Artifact-Tolerant Neural Recording Circuit," in 2022 IEEE International Symposium on Circuits and Systems (ISCAS), 2022.

[52] H.-G. Rhew et al., "A fully self-contained logarithmic closed-loop deep brain stimulation SoC with wireless telemetry and wireless power management," IEEE Journal of Solid-State Circuits, vol. 49, no. 10, pp. 2213-2227, 2014.

[53] A. L. Hodgkin and A. F. Huxley, "Propagation of electrical signals along giant nerve fibres," Proceedings of the Royal Society of London. Series B-Biological Sciences, vol. 140, no. 899, pp. 177-183, 1952.

[54] J. A. S. Kelso, Dynamic Patterns: The Self-Organization of Brain and Behavior. Cambridge, MA: MIT Press, 1995.

[55] E. P. Hoel, L. Albantakis, and G. Tononi, "Quantifying causal emergence shows that macro can beat micro," Proceedings of the National Academy of Sciences, vol. 110, no. 49, pp. 19790-19795, 2013.

[56] P. L. Nunez, Electric Fields of the Brain: The Neurophysics of EEG, 2nd ed. Oxford University Press, 2006.

[57] M. Breakspear et al., "A unifying explanation of primary generalized seizures through nonlinear brain modeling and bifurcation analysis," Cerebral Cortex, vol. 16, no. 9, pp. 1296-1313, 2006.

[58] J. A. Roberts et al., "Scale-free bursting in human cortex following hypoxia at birth," Journal of Neuroscience, vol. 34, no. 19, pp. 6557-6572, 2014.

[59] I. Bojak, Z. V. Stoyanov, and D. T. J. Liley, "Emergence of spatially heterogeneous burst suppression in a neural field model of electrocortical activity," Frontiers in Systems Neuroscience, vol. 9, pp. 18, 2015.

[60] A. J. K. Phillips and P. A. Robinson, "A quantitative model of sleep-wake dynamics based on the physiology of the brainstem ascending arousal system," Journal of Biological Rhythms, vol. 22, no. 2, pp. 167-179, 2007.

[61] I. Bojak and D. T. J. Liley, "Modeling the effects of anesthesia on the electroencephalogram," Physical Review E—Statistical, Nonlinear, and Soft Matter Physics, vol. 71, no. 4, pp. 041902, 2005.

[62] C. J. Honey et al., "Network structure of cerebral cortex shapes functional connectivity on multiple time scales," Proceedings of the National Academy of Sciences, vol. 104, no. 24, pp. 10240-10245, 2007.

[63] G. Deco et al., "Key role of coupling, delay, and noise in resting brain fluctuations," Proceedings of the National Academy of Sciences, vol. 106, no. 25, pp. 10302-10307, 2009.

[64] P. A. Robinson, "Dynamics of large scale brain activity in normal arousal states and epileptic seizures," Physical Review E, vol. 65, pp. 1-10, 2001.

[65] F. Freyer et al., "Biophysical mechanisms of multistability in resting-state cortical rhythms," Journal of Neuroscience, vol. 31, no. 17, pp. 6353-6361, 2011.

[66] A. Omurtag, B. W. Knight, and L. Sirovich, "On the simulation of large populations of neurons," Journal of Computational Neuroscience, vol. 8, pp. 51-63, 2000.

[67] N. Fourcaud and N. Brunel, "Dynamics of the firing probability of noisy integrate-and-fire neurons," Neural Computation, vol. 14, no. 9, pp. 2057-2110, 2002.

[68] S. El Boustani and A. Destexhe, "A master equation formalism for macroscopic modeling of asynchronous irregular activity states," Neural Computation, vol. 21, no. 1, pp. 46-100, 2009.

[69] G. Deco, C. J. G., and E. J. B., "The dynamic brain: from spiking neurons to neural masses and cortical fields," PLoS Comput. Biol., vol. 4, no. 8, pp. e1000092, Aug. 2008.

[70] L. M. Harrison, O. David, and K. J. Friston, "Stochastic models of neuronal dynamics," Philosophical Transactions of the Royal Society B: Biological Sciences, vol. 360, no. 1457, pp. 1075-1091, May 2005.

[71] W. J. Ma, J. W. Beck, and M. A. B. et al., "Bayesian inference with probabilistic population codes," Nature Neuroscience, vol. 9, no. 11, pp. 1432-1438, Nov. 2006.

[72] K. Friston, "The free-energy principle: a unified brain theory?," Nature Reviews Neuroscience, vol. 11, no. 2, pp. 127-138, Feb. 2010.

[73] B. H. Jansen and V. G. Rit, "Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns," Biological Cybernetics, vol. 73, no. 4, pp. 357-366, Oct. 1995.

[74] F. H. Lopes da Silva, H. R. Storm van Leeuwen, and G. G. G. et al., "Model of brain rhythmic activity: the alpha-rhythm of the thalamus," Kybernetik, vol. 15, pp. 27-37, 1974.

[75] O. C. J. Lippold, "Mass action in the nervous system—Examination of the neurophysiological basis of adaptive behaviour through the EEG: By W. J. Freeman, xx+ 489 pages, 191 illustrations, 18 tables, Academic Press, New York, San Francisco, London, 1975, US $34.50," 1977, pp. 146.

[76] R. H. Wilson, "Excitatory and inhibitory interactions in localized populations of neurons," Biophysical Journal, vol. 12, no. 1, pp. 153-170, Jan. 1972.

[77] R. A. Stefanescu and V. K. Jirsa, "Reduced representations of heterogeneous mixed neural networks with synaptic coupling," Physical Review E: Statistical, Nonlinear, and Soft Matter Physics, vol. 83, no. 2, pp. 026204, Feb. 2011.

[78] V. K. Jirsa and R. A. Stefanescu, "Neural population modes capture biologically realistic large scale network dynamics," Bulletin of Mathematical Biology, vol. 73, no. 2, pp. 325-343, Feb. 2011.

[79] P. Miller, and S. G. et al., "A recurrent network model of somatosensory parametric working memory in the prefrontal cortex," Cerebral Cortex, vol. 13, no. 11, pp. 1208-1218, Nov. 2003.

[80] S. Rich, J. L. H., and T. B. et al., "Neurostimulation stabilizes spiking neural networks by disrupting seizure-like oscillatory transitions," Scientific Reports, vol. 10, no. 1, pp. 15408, Oct. 2020.

[81] C. Cakan, N. Jajcay, and K. Obermayer, "neurolib: A simulation framework for whole-brain neural mass modeling," Cognitive Computation, vol. 15, no. 4, pp. 1132-1152, Aug. 2023.

[82] H. R. Wilson and J. D. Cowan, "A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue," Kybernetik, vol. 13, no. 2, pp. 55-80, 1973.

[83] R. M. Borisyuk and A. B. Kirillov, "Bifurcation analysis of a neural network model," Biological Cybernetics, vol. 66, no. 4, pp. 319-325, 1992.

[84] G. N. Borisyuk, R. M. Borisyuk, and A. B. Kirillov, "Dynamics and bifurcations of two coupled neural oscillators with different connection types," Bulletin of Mathematical Biology, vol. 57, no. 5, pp. 809-840, 1995.

[85] H. G. E. Meijer, J. M. C. and P. A. R., "Modeling focal epileptic activity in the Wilson–Cowan model with depolarization block," The Journal of Mathematical Neuroscience (JMN), vol. 5, pp. 1-17, 2015.

[86] K. Burrage, P. M. Burrage, and T. Tian, "Numerical methods for strong solutions of stochastic differential equations: an overview," Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, vol. 460, no. 2041, pp. 373-402, 2004.

[87] M. Breakspear, "Dynamic models of large-scale brain activity," Nature Neuroscience, vol. 20, no. 3, pp. 340-352, Mar. 2017.

[88] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," Physical Review, vol. 36, no. 5, pp. 823-841, Nov. 1930.

[89] A. Burrello, G. D. and N. S., "Hyperdimensional computing with local binary patterns: One-shot learning of seizure onset and identification of ictogenic brain regions using short-time iEEG recordings," IEEE Transactions on Biomedical Engineering, vol. 67, no. 2, pp. 601-613, Feb. 2019.

[90] S. Singer and J. Nelder, "Nelder-Mead algorithm," Scholarpedia, vol. 4, no. 7, pp. 2928, 2009.

[91] J. Žiburkus, J. R. Cressman, and S. J. Schiff, "Seizures as imbalanced up states: excitatory and inhibitory conductances during seizure-like events," Journal of Neurophysiology, vol. 109, no. 5, pp. 1296-1306, May 2013.

[92] K. Chen, R. G. and B. T., "Persistently modified h-channels after complex febrile seizures convert the seizure-induced enhancement of inhibition to hyperexcitability," Nature Medicine, vol. 7, no. 3, pp. 331-337, Mar. 2001.

[93] D. E. Naylor, "Glutamate and GABA in the balance: convergent pathways sustain seizures during status epilepticus," Epilepsia, vol. 51, no. 1, pp. 106-109, Jan. 2010.

[94] A. Nabi and J. Moehlis, "Single input optimal control for globally coupled neuron networks," Journal of Neural Engineering, vol. 8, no. 6, pp. 065008, Dec. 2011.

[95] J. Li, I. Dasanayake, and J. Ruths, "Control and synchronization of neuron ensembles," IEEE Transactions on Automatic Control, vol. 58, no. 8, pp. 1919-1930, Aug. 2013.

[96] O. V. Popovych and P. A. Tass, "Control of abnormal synchronization in neurological disorders," Frontiers in Neurology, vol. 5, pp. 268, Oct. 2014.

[97] X. Chen, Y. L. and H. M., "Machine and cognitive intelligence for human health: systematic review," Brain Informatics, vol. 9, no. 1, pp. 5, Jan. 2022.

[98] S.-F. Liang, H.-C. Wang, and W.-L. Chang, "Combination of EEG complexity and spectral analysis for epilepsy diagnosis and seizure detection," EURASIP Journal on Advances in Signal Processing, vol. 2010, pp. 1-15, 2010.

[99] M. A. B. Altaf and J. Yoo, "A 1.83 μJ/classification, 8-channel, patient-specific epileptic seizure classification SoC using a non-linear support vector machine," IEEE Transactions on Biomedical Circuits and Systems, vol. 10, no. 1, pp. 1-11, Feb. 2016.

[100] M. R. Karimi and H. Kassiri, "A multi-feature nonlinear-SVM seizure detection algorithm with patient-specific channel selection and feature customization," in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2020, pp. 1-5.

[101] T. Zhan, S. Guraya, and H. Kassiri, "A resource-optimized VLSI architecture for patient-specific seizure detection using frontal-lobe EEG," in 2019 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2019, pp. 1-5.

[102] A. Dzielinski and D. Sierociuk, "Adaptive feedback control of fractional order discrete state-space systems," in International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), vol. 1, IEEE, 2005, pp. 124-129.

[103] E. F. Camacho and C. Bordons, Model Predictive Control, Springer Science & Business Media, 2013.

[104] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," Automatica, vol. 36, no. 6, pp. 789-814, Jun. 2000.

[105] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," Control Engineering Practice, vol. 11, no. 7, pp. 733-764, Jul. 2003.

[106] F. Allgöwer and A. Zheng, Eds., Nonlinear Model Predictive Control, vol. 26, Birkhäuser, 2012.

[107] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," Automatica, vol. 25, no. 3, pp. 335-348, Mar. 1989.

[108] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," Computers & Chemical Engineering, vol. 23, no. 4-5, pp. 667-682, Apr. 1999.

[109] S. Chatterjee, M. A., and R. R., "Fractional-order model predictive control as a framework for electrical neurostimulation in epilepsy," Journal of Neural Engineering, vol. 17, no. 6, 066, Dec. 2020.

[110] S. Chang, et al., "Model predictive control for seizure suppression based on nonlinear auto-regressive moving-average Volterra model," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 28, no. 10, pp. 2173-2183, 2020.

[111] E. Hazan, K. Singh, and C. Zhang, "Learning linear dynamical systems via spectral filtering," Advances in Neural Information Processing Systems 30, 2017.

[112] N. Lanzetti, et al., "Recurrent neural network based MPC for process industries," 2019 18th European Control Conference (ECC), IEEE, 2019.

[113] K. Bieker, et al., "Deep model predictive flow control with limited sensor data and online learning," Theoretical and Computational Fluid Dynamics, vol. 34, no. 4, pp. 577-591, 2020.

[114] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," Proceedings of the National Academy of Sciences, vol. 17, no. 5, pp. 315-318, 1931.

[115] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," Journal of Fluid Mechanics, vol. 656, pp. 5-28, 2010.

[116] I. Kevrekidis, C. W. Rowley, and M. Williams, "A kernel-based method for data-driven Koopman spectral analysis," Journal of Computational Dynamics, vol. 2, no. 2, pp. 247-265, 2016.

[117] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," Journal of Nonlinear Science, vol. 25, pp. 1307-1346, 2015.

[118] X. Zhang, et al., "Robust learning-based predictive control for discrete-time nonlinear systems with unknown dynamics and state constraints," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 52, no. 12, pp. 7314-7327, 2022.

[119] X. Zhang, et al., "Robust tube-based model predictive control with Koopman operators," Automatica, vol. 137, 2022, Art. no. 110114.

[120] Y. Xiao, et al., "Deep neural networks with Koopman operators for modeling and control of autonomous vehicles," IEEE Transactions on Intelligent Vehicles, vol. 8, no. 1, pp. 135-146, 2022.

[121] S. E. Otto and C. W. Rowley, "Linearly recurrent autoencoder networks for learning dynamics," SIAM Journal on Applied Dynamical Systems, vol. 18, no. 1, pp. 558-593, 2019.

[122] J. Morton, F. D. Witherden, and M. J. Kochenderfer, "Deep variational Koopman models: Inferring Koopman observations for uncertainty-aware dynamics modeling and control," arXiv preprint arXiv:1902.09742, 2019.

[123] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," Automatica, vol. 93, pp. 149-160, 2018.

[124] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010.

[125] D. P. Kingma, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[126] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, Cambridge, MA: MIT Press, 2016. Available: http://www.deeplearningbook.org

[127] M. Abadi, et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," arXiv preprint arXiv:1603.04467, 2016

Appendix A Wilson-Cowan Model Python Implementation

A.1 WCModel Class

```
from . import loadDefaultParams as dp  # Import default parameter loading function

from . import timeIntegration as ti  # Import time integration function

from ..model import Model  # Import base Model class

class WCModel(Model):

    name = "wc"  # Model name

    description = "Wilson-Cowan model"  # Model description

    init_vars = ["exc_init", "inh_init", "exc_ou", "inh_ou"]  # Initial variables

    state_vars = ["exc", "inh", "exc_ou", "inh_ou"]  # State variables

    output_vars = ["exc", "inh"]  # Output variables

    default_output = "exc"  # Default output variable

    input_vars = ["exc_ext", "inh_ext"]  # Input variables

    default_input = "exc_ext"  # Default input variable

    # Transform input to the BOLD model

    boldInputTransform = lambda self, x: x * 50

    def __init__(self, params=None, Cmat=None, Dmat=None, seed=None):

        self.Cmat = Cmat  # Connectivity matrix
```

```
        self.Dmat = Dmat  # Delay matrix
        self.seed = seed  # Seed for random number generation
        # Set the integration function
        integration = ti.timeIntegration
        # Load default parameters if not provided
        if params is None:
                params = dp.loadDefaultParams(Cmat=self.Cmat, Dmat=self.Dmat,
seed=self.seed)
                 # Initialize the base Model class
                super().__init__(integration=integration, params=params)
```

A.2 Loading NMM Parameters

```
import numpy as np
from ...utils.collections import dotdict
def loadDefaultParams(Cmat=None, Dmat=None, seed=None):
        params = dotdict({})
    ### runtime parameters
    params.dt = 0.1  # ms 0.1ms is reasonable
    params.duration = 2000  # Simulation duration (ms)
    np.random.seed(seed)  # seed for RNG of noise and ICs
```

```
params.seed = seed

# ------------------------------------------------------------------------

# global whole-brain network parameters

# ------------------------------------------------------------------------

params.signalV = 20.0  # signal transmission speed between areas

params.K_gl = 0.6  # global coupling strength

if Cmat is None:

    params.N = 1

    params.Cmat = np.zeros((1, 1))

    params.lengthMat = np.zeros((1, 1))

else:

    params.Cmat = Cmat.copy()  # coupling matrix

    np.fill_diagonal(params.Cmat, 0)  # no self connections

    params.N = len(params.Cmat)  # number of nodes

    params.lengthMat = Dmat

# ------------------------------------------------------------------------

# local node parameters

# ------------------------------------------------------------------------

params.tau_ou = 5.0  # ms Timescale of the Ornstein-Uhlenbeck noise process
```

```
params.sigma_ou = 0.0  # noise intensity

params.exc_ou_mean = 0.0  # OU process mean

params.inh_ou_mean = 0.0  # OU process mean

params.stim = 0

params.tau_exc = 2.5  # excitatory time constant

params.tau_inh = 3.75  # inhibitory time constant

params.c_excexc = 16  # local E-E coupling

params.c_excinh = 15  # local E-I coupling

params.c_inhexc = 12  # local I-E coupling

params.c_inhinh = 3  # local I-I coupling

params.a_exc = 1.5  # excitatory gain

params.a_inh = 1.5  # inhibitory gain

params.mu_exc = 3.0  # excitatory firing threshold

params.mu_inh = 3.0  # inhibitory firing threshold

params.B = 3

params.alpha = 0.33

params.exc_ext = 0  # baseline external input to E

params.inh_ext = 0  # baseline external input to I

# ------------------------------------------------------------------------
```

```
    params.exc_init = 0.05 * np.random.uniform(0, 1, (params.N, 1))
    params.inh_init = 0.05 * np.random.uniform(0, 1, (params.N, 1))
    # Ornstein-Uhlenbeck noise state variables
    params.exc_ou = np.zeros((params.N,))
    params.inh_ou = np.zeros((params.N,))
    return params
def computeDelayMatrix(lengthMat, signalV, segmentLength=1):
    normalizedLenMat = lengthMat * segmentLength
    # Interareal connection delays, Dmat(i,j) in ms
    if signalV > 0:
        Dmat = normalizedLenMat / signalV
    else:
        Dmat = lengthMat * 0.0
    return Dmat
```

A.3 Time Integration

```
import numpy as np
import numba
from . import loadDefaultParams as dp
from ...utils import model_utils as mu
```

```
import numpy as np
import matplotlib.pyplot as plt
from . import library_meow as lib
def timeIntegration(params):
    dt = params["dt"]  # Time step for the Euler integration (ms)
    duration = params["duration"]  # Simulation duration (ms)
    RNGseed = params["seed"]  # Seed for RNG
    # Local parameters
    tau_exc = params["tau_exc"]
    tau_inh = params["tau_inh"]
    c_excexc = params["c_excexc"]
    c_excinh = params["c_excinh"]
    c_inhexc = params["c_inhexc"]
    c_inhinh = params["c_inhinh"]
    a_exc = params["a_exc"]
    a_inh = params["a_inh"]
    mu_exc = params["mu_exc"]
    mu_inh = params["mu_inh"]
    B, alpha = params["B"], params["alpha"]
```

```
# External input parameters (Ornstein-Uhlenbeck process)
tau_ou = params["tau_ou"]
sigma_ou = params["sigma_ou"]
exc_ou_mean = params["exc_ou_mean"]
inh_ou_mean = params["inh_ou_mean"]
# Global coupling parameters
Cmat = params["Cmat"]  # Connectivity matrix
N = len(Cmat)  # Number of nodes
K_gl = params["K_gl"]  # Global coupling strength
lengthMat = params["lengthMat"]
signalV = params["signalV"]
# Compute delay matrix
if N == 1:
    Dmat = np.zeros((N, N))
else:
    Dmat = dp.computeDelayMatrix(lengthMat, signalV)
    Dmat[np.eye(len(Dmat)) == 1] = np.zeros(len(Dmat))
Dmat_ndt = np.around(Dmat / dt).astype(int)  # Delay matrix in multiples of dt
params["Dmat_ndt"] = Dmat_ndt
```

```
# Initialization

t = np.arange(1, round(duration, 6) / dt + 1) * dt  # Time variable (ms)

sqrt_dt = np.sqrt(dt)

max_global_delay = np.max(Dmat_ndt)

startind = int(max_global_delay + 1)  # Timestep to start integration at

exc_ou = params["exc_ou"]

inh_ou = params["inh_ou"]

excs = np.zeros((N, startind + len(t)))

inhs = np.zeros((N, startind + len(t)))

stim = params["stim"]

exc_ext = mu.adjustArrayShape(params["exc_ext"], excs)

inh_ext = mu.adjustArrayShape(params["inh_ext"], inhs)

# Set initial values

if np.shape(params["exc_init"])[1] == 1:

    exc_init = np.dot(params["exc_init"], np.ones((1, startind)))

    inh_init = np.dot(params["inh_init"], np.ones((1, startind)))

else:

    exc_init = params["exc_init"][:, -startind:]

    inh_init = params["inh_init"][:, -startind:]
```

```
    exc_input_d = np.zeros(N)  # Delayed input to exc

    inh_input_d = np.zeros(N)  # Delayed input to inh

    np.random.seed(RNGseed)

    excs[:, startind:] = np.random.standard_normal((N, len(t)))

    inhs[:, startind:] = np.random.standard_normal((N, len(t)))

    excs[:, :startind] = exc_init

    inhs[:, :startind] = inh_init

    noise_exc = np.zeros((N,))

    noise_inh = np.zeros((N,))

    return timeIntegration_njit_elementwise(startind, t, dt, sqrt_dt, N, Cmat, K_gl, Dmat_ndt,
excs, inhs, exc_input_d, inh_input_d, exc_ext, inh_ext, tau_exc, tau_inh, a_exc, a_inh,
mu_exc, mu_inh, c_excexc, c_excinh, c_inhexc, c_inhinh, noise_exc, noise_inh, exc_ou,
inh_ou, exc_ou_mean, inh_ou_mean, tau_ou, sigma_ou, B, alpha, stim)

@numba.njit

def timeIntegration_njit_elementwise(startind, t, dt, sqrt_dt, N, Cmat, K_gl, Dmat_ndt, excs,
inhs, exc_input_d, inh_input_d, exc_ext, inh_ext, tau_exc, tau_inh, a_exc, a_inh, mu_exc,
mu_inh, c_excexc, c_excinh, c_inhexc, c_inhinh, noise_exc, noise_inh, exc_ou, inh_ou,
exc_ou_mean, inh_ou_mean, tau_ou, sigma_ou, B, alpha, stim)

    ### Integrate ODE system:

    def S_E(x):

        return 1.0 / (1.0 + np.exp(-a_exc * (x - mu_exc))) - 1.0 / (1.0 + np.exp(a_exc * mu_exc))

    def S_I(x):
```

```
    return 1.0 / (1.0 + np.exp(-a_inh * (x - mu_inh))) - 1.0 / (1.0 + np.exp(a_inh * mu_inh))

B0 = 0

k = 1

for i in range(startind, startind + len(t)):

    if i < 5 * 1024:

        B0 = 1

    elif tau_exc == 20:

        B0 = 6

    else:

        B0 = 1

    # Loop through all the nodes

    for no in range(N):

        noise_exc[no] = excs[no, i]  # Noise saved in the activity array

        noise_inh[no] = inhs[no, i]

        # Delayed input to each node

        exc_input_d[no] = 0

        if no == 0:

            exc_input_d[no] = excs[1, i - 2]

        elif no == N - 1:
```

```
    exc_input_d[no] = excs[N - 2, i - 2]
else:
    exc_input_d[no] = excs[no - 1, i - 2] + excs[no + 1, i - 2]
# Wilson-Cowan model
exc_rhs = (
    1 / tau_exc
    * (
        -excs[no, i - 1]
        + (1 - excs[no, i - 1])
        * S_E(
            c_excexc * excs[no, i - 1]
            - c_excinh * inhs[no, i - 1]
            + c_excexc * alpha * exc_input_d[no]
            + B0
        )
        + exc_ou[no]
    )
)
inh_rhs = (
```

```
    1 / tau_inh
    * (
        -inhs[no, i - 1]
        + (1 - inhs[no, i - 1])
        * S_I(
            c_excinh * excs[no, i - 1]
            - c_inhinh * inhs[no, i - 1]
            + stim[i]
        )
        + inh_ou[no]
    )
)
# Euler integration
excs[no, i] = excs[no, i - 1] + dt * exc_rhs
inhs[no, i] = inhs[no, i - 1] + dt * inh_rhs
excs[no, i] = excs[no, i] * k
inhs[no, i] = inhs[no, i] * k
# Ornstein-Uhlenbeck process
exc_ou[no] = (
```

```
            exc_ou[no] + (exc_ou_mean - exc_ou[no]) * dt / tau_ou + sigma_ou * sqrt_dt *
noise_exc[no]

        )

        inh_ou[no] = (

            inh_ou[no] + (inh_ou_mean - inh_ou[no]) * dt / tau_ou + sigma_ou * sqrt_dt *
noise_inh[no]

        )

    return t, excs, inhs, exc_ou, inh_ou
```

Appendix B Python Code for Patient-Specific Optimization of NMM

B.1 Library: Functions Used in the Main Optimization Code

```
# OPEN MAT FILE

def open_matfile(Patient, ID ):

    if Patient<9:

        folder_path = "/local/data0/Rojin/codes/ID0%d_%dh.mat"%(Patient,ID)

    else:

       folder_path="/local/data0/Rojin/codes/ID%d_%dh.mat"%(Patient,ID)


    mat = scipy.io.loadmat(folder_path)

    keysList = list(mat.keys())

    electrodes=mat['EEG']

    return  electrodes

#Removing DC Level

def make_mean_zero(signal):

    mean_value = np.mean(signal)

    return signal - mean_value

# Generate Stimulation Pulse

def generate_pulse_signal(duration, freq, duty_cycle):
```

```
    t = np.linspace(0, duration, int(duration * 1024), endpoint=False)

    cycles = t * freq

    pulse_signal = 1 * (1 + scipy_signal.square(2 * np.pi * cycles, duty=duty_cycle/100))-1

    #pulse_signal = make_mean_zero(pulse_signal)

    return pulse_signal

def make_mean_zero(signal):

    return signal - np.mean(signal)

#Set Parameters of NMM

def set_NMM_model_f (tau_exc,tau_inh,c_excexc,c_excinh,c_inhexc,c_inhinh,a_exc,a_inh,mu_exc,mu_inh,B,alpha, amp,freq,duty_cycle):

    from neurolib_modified_stimulation.models.wc import WCModel

    #stim = generate_pulse_signal(20, freq,duty_cycle) * amp

    stim = generate_pulse_signal(20, freq,duty_cycle) * amp

    stim [:1024*10] = stim [:1024*10]*0

    stim [1024*15:] = stim [1024*15:]*0

    w = 1

    c_matrix = c_matrix = np.random.rand(3, 3)

    d = 0

    d_matrix = c_matrix = np.random.rand(3, 3)
```

```
model = WCModel(Cmat=c_matrix, Dmat=d_matrix)

model = WCModel(Cmat=c_matrix, Dmat=d_matrix)

model.params['duration'] = 1024*15

model.params['dt'] = 1000/1024

model.params['stim'] = stim

model.params['sigma_ou'] = 0.01

model.params["tau_exc"]  = tau_exc

model.params["tau_inh"]  = tau_inh

model.params["c_excexc"] = c_excexc

model.params["c_excinh"] = c_excinh

model.params["c_inhexc"] = c_inhexc

model.params["c_inhinh"] = c_inhinh

model.params["a_exc"]    = a_exc

model.params["a_inh"]    = a_inh

model.params["mu_exc"]   = mu_exc

model.params["mu_inh"]   = mu_inh

model.params["B"], model.params["alpha"] = B,alpha

model.run()

signals_exc = model.outputs["exc"]
```

```
    signals_inh = model.outputs["inh"]

    signal_EEG = []

    for t in range(len(signals_exc[0])):

        J = 0

        for i in range(1):

            J = J+ c_excexc * signals_exc[i][t] - c_excinh* signals_inh[i][t] + c_excexc*0.1*(signals_exc[i-1][t]+signals_exc[i+1][t])+B

        signal_EEG.append(J)

    signal_EEG = np.array(signal_EEG)

    return signals_exc,signals_inh,signal_EEG,stim

#FFT

def FFT(ls,freq):

    N=len(ls)

    T=1/freq

    xf = fftfreq(N, T)[:N//2]

    yf = fft(ls)

    ff_t=2.0/N * np.abs(yf[0:N//2])

    le=len(xf)

    return xf,ff_t
```

B.2 Main Optimization Code

```
import library as lib

import numpy as np

from scipy.optimize import minimize

import matplotlib.pyplot as plt

initial_guess = [

                100.0,    #'tau_exc0'

                100.0,   #'tau_inh0'

                16.0,   #'c_excexc0'

                18.0,   #'c_excinh0'

                12.0,   #'c_inhexc0'

                3.0,    #'c_inhinh0'

                1.5828,    #'a_exc0'

                2.2201,    #'a_inh0'

                5.2516,    #'mu_exc0'

                3.7512,    #'mu_inh0'

                20,      #B

                0.1,    #alpha

                50,   #amp

                30,  #freq

                50,  #duty_cycle]
```

```
signal = []
for t in [20,30,40,50,60,70,80,90,100,120,140]:
    initial_guess[0] = t
    initial_guess[1] = t
    signals_exc,signals_inh,signal_NMM,stim = lib.set_NMM_model_f(*initial_guess)
    signal.append(signal_NMM)
electrodes = lib.open_matfile(10,6)
signal_ref = electrodes[0][1024*100:1024*101]
xf,fft_ref = lib.FFT(signal_ref,1024)
amps = [1 for i in range(11)]
def cost_function(amps):
    signal_NMM = np.zeros(len(signal[0][:1024*1]))
    for i in range(len(signal)):
        signal_NMM = signal_NMM+amps[i]*signal[i][:1024*1]
    xf,fft_NMM = lib.FFT(signal_NMM,1024)
    error = np.mean((np.array(fft_NMM[1:]) - np.array(fft_ref[1:]))**2)
    return error
result = minimize(cost_function, amps, method='Nelder-Mead')
min_value = result.fun
amps = result.x
signal_NMM = np.zeros(1024*15)
for i in range(len(signal)):
```

```
    signal_NMM = signal_NMM+amps[i]*signal[i][:1024*15]
```

B.3 Stimulation Efficacy Using Discrete Frequencies

```
Amp = 0.05

Freq = 10

 initial_guess = [

                        100.0,    #'tau_exc0'

                        100.0,   #'tau_inh0'

                        16.0,   #'c_excexc0'

                        18.0,   #'c_excinh0'

                        12.0,   #'c_inhexc0'

                        3.0,    #'c_inhinh0'

                        1.5828,    #'a_exc0'

                        2.2201,    #'a_inh0'

                        5.2516,    #'mu_exc0'

                        3.7512,    #'mu_inh0'

                        20,      #B

                        0.1,    #alpha

                        amp,   #amplitude
```

```
                freq,  #frequency

                50,  #duty_cycle

signal = []

for t in [20,30,40,50,60,70,80,90,100,120,140]:

initial_guess[0] = t

initial_guess[1] = t

signals_exc,signals_inh,signal_NMM,stim = lib.set_NMM_model_f(*initial_guess)

signal.append(signal_NMM)

signal_NMM = np.zeros(1024*15)

for i in range(len(signal)):

        signal_NMM = signal_NMM+amps[i]*signal[i][:1024*15]

        xf,energy_value_normal = lib.FFT(signal_NMM[4*1024:5*1024],1024)

        xf,energy_value_seizur = lib.FFT(signal_NMM[9*1024:10*1024],1024)

        xf,energy_value_stimul = lib.FFT(signal_NMM[14*1024:15*1024],1024)

       cost_value=
(lib.pearson_corr(energy_value_stimul[1:50],energy_value_seizur[1:50])/1000*lib.pearson_co
rr(energy_value_stimul[1:50],energy_value_normal[1:50]))
```

B.4 Stimulation Efficacy Using Spectral Bands

B.4.1 Calculated with Ideal Filter

```
initial_guess = [
                100.0,    #'tau_exc0'
                100.0,   #'tau_inh0'
                16.0,   #'c_excexc0'
                18.0,   #'c_excinh0'
                12.0,   #'c_inhexc0'
                3.0,    #'c_inhinh0'
                1.5828,    #'a_exc0'
                2.2201,    #'a_inh0'
                5.2516,    #'mu_exc0'
                3.7512,    #'mu_inh0'
                20,      #B
                0.1,    #alpha
                amp,   #amp
                freq,  #freq
                50,  #duty_cycle]
```

```
        signal = []
        for t in [20,30,40,50,60,70,80,90,100,120,140]:
            initial_guess[0] = t
            initial_guess[1] = t
            signals_exc,signals_inh,signal_NMM,stim = lib.set_NMM_model_f(*initial_guess)
            signal.append(signal_NMM)
        signal_NMM = np.zeros(1024*15)
        for i in range(len(signal)):
            signal_NMM = signal_NMM+amps[i]*signal[i][:1024*15]
        xf,energy_value_normal = lib.FFT(signal_NMM[4*1024:5*1024],1024)
        xf,energy_value_seizur = lib.FFT(signal_NMM[9*1024:10*1024],1024)
        xf,energy_value_stimul = lib.FFT(signal_NMM[14*1024:15*1024],1024)
        cost_value =
(lib.pearson_corr(energy_value_stimul[1:50],energy_value_seizur[1:50])/1000*lib.pearson_co
rr(energy_value_stimul[1:50],energy_value_normal[1:50]))
```

B.4.2 Calculated with FIR filters

```
#Designing Filters
fs = 1024  # Sampling rate
attenuation_dB = 10  # Desired attenuation in the stopband in dB
```

```
transition_width = 4 # Transition width in Hz

low_cutoff_hz = 0.10  # Lower cutoff frequency for bandpass filter

high_cutoff_hz = 5  # Upper cutoff frequency for bandpass filter

numtaps, beta = signal.kaiserord(attenuation_dB, transition_width / (0.5 * fs))

fir_coefficients1 = signal.firwin(numtaps, [low_cutoff_hz, high_cutoff_hz], window=('kaiser',
beta), pass_zero=False, fs=fs)

np.savetxt('fir_coefficients1.txt', fir_coefficients1)

low_cutoff_hz = 5  # Lower cutoff frequency for bandpass filter

high_cutoff_hz = 10  # Upper cutoff frequency for bandpass filter

numtaps, beta = signal.kaiserord(attenuation_dB, transition_width / (0.5 * fs))

fir_coefficients2 = signal.firwin(numtaps, [low_cutoff_hz, high_cutoff_hz], window=('kaiser',
beta), pass_zero=False, fs=fs)

np.savetxt('fir_coefficients2.txt', fir_coefficients2)

low_cutoff_hz = 10  # Lower cutoff frequency for bandpass filter

high_cutoff_hz = 15  # Upper cutoff frequency for bandpass filter

numtaps, beta = signal.kaiserord(attenuation_dB, transition_width / (0.5 * fs))

fir_coefficients3 = signal.firwin(numtaps, [low_cutoff_hz, high_cutoff_hz], window=('kaiser',
beta), pass_zero=False, fs=fs)

np.savetxt('fir_coefficients3.txt', fir_coefficients3)

low_cutoff_hz = 15  # Lower cutoff frequency for bandpass filter

high_cutoff_hz = 20  # Upper cutoff frequency for bandpass filter
```

```
numtaps, beta = signal.kaiserord(attenuation_dB, transition_width / (0.5 * fs))

fir_coefficients4 = signal.firwin(numtaps, [low_cutoff_hz, high_cutoff_hz], window=('kaiser', beta), pass_zero=False, fs=fs)

np.savetxt('fir_coefficients4.txt', fir_coefficients4)

low_cutoff_hz = 20  # Lower cutoff frequency for bandpass filter

high_cutoff_hz = 25  # Upper cutoff frequency for bandpass filter

numtaps, beta = signal.kaiserord(attenuation_dB, transition_width / (0.5 * fs))

fir_coefficients5 = signal.firwin(numtaps, [low_cutoff_hz, high_cutoff_hz], window=('kaiser', beta), pass_zero=False, fs=fs)

np.savetxt('fir_coefficients5.txt', fir_coefficients5)

low_cutoff_hz = 25  # Lower cutoff frequency for bandpass filter

high_cutoff_hz = 30  # Upper cutoff frequency for bandpass filter

numtaps, beta = signal.kaiserord(attenuation_dB, transition_width / (0.5 * fs))

fir_coefficients6 = signal.firwin(numtaps, [low_cutoff_hz, high_cutoff_hz], window=('kaiser', beta), pass_zero=False, fs=fs)

np.savetxt('fir_coefficients6.txt', fir_coefficients5)

def remove_dc_level(signal):

    dc_level = np.mean(signal)

    return signal - dc_level# Plot the frequency response of the filter

def calculateenergyforsignalFIR(eeg_signal):
```

```
    energy_values = [np.sum((signal.lfilter(fir_coefficients1, 1.0,
remove_dc_level(eeg_signal)))**2)*1000,np.sum((signal.lfilter(fir_coefficients2, 1.0,
remove_dc_level(eeg_signal)))**2)*1000, np.sum((signal.lfilter(fir_coefficients3, 1.0,
remove_dc_level(eeg_signal)))**2)*1000, np.sum((signal.lfilter(fir_coefficients4, 1.0,
remove_dc_level(eeg_signal)))**2)*1000, np.sum((signal.lfilter(fir_coefficients5, 1.0,
remove_dc_level(eeg_signal)))**2)*1000, np.sum((signal.lfilter(fir_coefficients6, 1.0,
remove_dc_level(eeg_signal)))**2)*1000]

    return energy_values

initial_guess = [

                100.0,    #'tau_exc0'

                100.0,   #'tau_inh0'

                16.0,   #'c_excexc0'

                18.0,   #'c_excinh0'

                12.0,   #'c_inhexc0'

                3.0,    #'c_inhinh0'

                1.5828,    #'a_exc0'

                2.2201,    #'a_inh0'

                5.2516,    #'mu_exc0'

                3.7512,    #'mu_inh0'

                20,      #B

                0.1,    #alpha

                amp,   #amp
```

```
            freq,  #freq
            50,  #duty_cycle]
signall = []
for t in [20,30,40,50,60,70,80,90,100,120,140]:
    initial_guess[0] = t
    initial_guess[1] = t
    signals_exc,signals_inh,signal_NMM,stim = lib.set_NMM_model_f(*initial_guess)
    signall.append(signal_NMM)
signal_NMM = np.zeros(1024*15)
for i in range(len(signall)):
    signal_NMM = signal_NMM+amps[i]*signall[i][:1024*15]
energy_value_normal = calculateenergyforsignalFIR (signal_NMM[4*1024:5*1024]/1000)
energy_value_seizur = calculateenergyforsignalFIR (signal_NMM[9*1024:10*1024]/1000)
energy_value_stimul = calculateenergyforsignalFIR (signal_NMM[14*1024:15*1024]/1000)
weights = []
for f in range(6):
    weights.append(abs(energy_value_seizur[f]-energy_value_normal[f]))
weights = normalize_to_range(weights)
cost_value = 0
```

```
for f in range(6):
    cost_value += (weights[f])*(energy_value_stimul[f])
```

B.5 Creating Dataset with Optimized NMM

```
import random
def creating_dataset():
    X_data = []
    C_data = []
    Y_data = []
    N_data = []
    freq_cost = []
    for i in range(88):
        amp = random.uniform(0.0, 5),   #amp
        freq = random.uniform(0.1,200),  #freq
        initial_guess = [
                        100.0,    #'tau_exc0'
                        100.0,   #'tau_inh0'
                        16.0,   #'c_excexc0'
                        18.0,   #'c_excinh0'
```

```
                12.0,   #'c_inhexc0'
                3.0,    #'c_inhinh0'
                1.5828,    #'a_exc0'
                2.2201,    #'a_inh0'
                5.2516,    #'mu_exc0'
                3.7512,    #'mu_inh0'
                20,      #B
                0.1,    #alpha
                amp, #amplitude
                freq, #frequency
                50,  #duty_cycle]
signall = []
for t in [20,30,40,50,60,70,80,90,100,120,140]:
    initial_guess[0] = t
    initial_guess[1] = t
    signals_exc,signals_inh,signal_NMM,stim = lib.set_NMM_model_f(*initial_guess)
    signall.append(signal_NMM)
signal_NMM = np.zeros(1024*15)
for i in range(len(signall)):
```

```
            signal_NMM = signal_NMM+amps[i]*signall[i][:1024*15]
        energy_value_normal = calculate_energy_for_signal(signal_NMM[4*1024:5*1024]/1000)
        energy_value_seizur = calculate_energy_for_signal(signal_NMM[9*1024:10*1024]/1000)
        energy_value_stimul =
calculate_energy_for_signal(signal_NMM[14*1024:15*1024]/1000)
        weights = []
        for f in range(6):
            weights.append(abs(energy_value_seizur[f]-energy_value_normal[f]))
        weights = normalize_to_range(weights)
        cost_value = 0
        for f in range(6):
            cost_value += (weights[f])*(energy_value_stimul[f])
        N_data.append(energy_value_normal)
        X_data.append(energy_value_seizur)
        C_data.append([amp,freq])
        Y_data.append(energy_value_stimul)
return N_data,X_data,C_data,Y_data
i = 0
N_data,X_data,C_data,Y_data,freq_cost = creating_dataset()
N_data = np.array(N_data)
```

```
X_data = np.array(X_data)

C_data = np.array(C_data)

Y_data = np.array(Y_data)

np.save('N_data%d.npy'%i, N_data)

np.save('X_data%d.npy'%i, X_data)

np.save('C_data%d.npy'%i, C_data)

np.save('Y_data%d.npy'%i, Y_data)
```

Appendix C Python Code for Patient-Optimized Adaptive Controller

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense

from tensorflow.keras.optimizers import Adam

from sklearn.model_selection import train_test_split

#Import dataset

def normalize_to_max_one(*datasets, threshold=None):

    global_max = max(np.max(data) for data in datasets)
```

```
    global_min = min(np.min(data) for data in datasets)

    normalized_datasets = [((data - global_min) / (global_max - global_min)) for data in
datasets]

    return normalized_datasets, global_max, global_min

N_data_t = np.load('N_data.npy')

X_data_t = np.load('X_data.npy')

Y_data_t = np.load('Y_data.npy')

C_data_t = np.load('C_data.npy')

N_data = np.array(N_data)

X_data = np.array(X_data)

C_data = np.array(C_data)

Y_data = np.array(Y_data)

N_data, X_data, Y_data = normalize_to_max_one(N_data, X_data, Y_data)

X_train, X_temp, C_train, C_temp, Y_train, Y_temp = train_test_split(X_data, C_data,
Y_data, test_size=0.5)

X_val, X_test, C_val, C_test, Y_val, Y_test = train_test_split(X_temp, C_temp, Y_temp,
test_size=0.5)

def build_autoencoder(input_dim, control_dim, encoding_dim):

    import tensorflow as tf

    from tensorflow.keras.optimizers import Adam

    from tensorflow.keras.initializers import RandomUniform
```

```
from tensorflow.keras.optimizers import RMSprop


# Define the Encoder
encoder = tf.keras.Sequential([
    tf.keras.layers.Dense(encoding_dim, activation='relu', input_shape=(input_dim,)),
    tf.keras.layers.Dense(encoding_dim, activation='relu'),
    tf.keras.layers.Dense(encoding_dim, activation='relu'),
    tf.keras.layers.Dense(encoding_dim, activation='relu'),
    tf.keras.layers.Dense(encoding_dim, activation='relu')
])
# Define the One-to-One Neural Networks (Identity function)
A_nn = tf.keras.layers.Dense(encoding_dim, activation=None, use_bias=False)
B_nn = tf.keras.layers.Dense(encoding_dim, activation=None, use_bias=False)
# Define the Decoder with positive initialization and ReLU activation
decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(encoding_dim, activation='relu', input_shape=(encoding_dim,)),
    tf.keras.layers.Dense(encoding_dim, activation='relu'),
    tf.keras.layers.Dense(encoding_dim, activation='relu'),
    tf.keras.layers.Dense(encoding_dim, activation='relu'),
```

```
        tf.keras.layers.Dense(input_dim, activation='relu'),
    ])
    # Define the input layers
    input_data = tf.keras.layers.Input(shape=(input_dim,))
    control_signal = tf.keras.layers.Input(shape=(control_dim,))
    # Connect the Encoder, One-to-One NN, and Decoder
    encoded_data = encoder(input_data)
    encoded_data = A_nn(encoded_data)
    control_signal_transformed = B_nn(control_signal)
    concatenated_data = tf.keras.layers.Add()([encoded_data, control_signal_transformed])
    decoded_data = decoder(concatenated_data)
    # Define the model
    model = tf.keras.Model(inputs=[input_data, control_signal], outputs=decoded_data)
    optimizer = RMSprop(learning_rate=0.0001)
    # Compile the model
    model.compile(optimizer=optimizer, loss='mse')
    return model, encoder, A_nn, B_nn, decoder
input_dim = 6
control_dim = 2
```

```
encoding_dim = 20

model, encoder, A_nn, B_nn, decoder = build_autoencoder(input_dim, control_dim,
encoding_dim)

# Train the model

history = model.fit(

    x=[X_train, C_train],  # Provide input data and control signals

    y=Y_train,  # Provide the corresponding labels

    epochs=800,  # Choose the number of epochs

    batch_size=700,  # Choose batch size

    validation_data=([X_val, C_val], Y_val)  # Optional: Provide validation data

)

def cost_function(amplitude, frequency):

    # Convert amplitude and frequency to numpy arrays if they aren't already

    amplitude = np.array([amplitude])

    frequency = np.array([frequency])

    # Placeholder for the actual model prediction

    energy_value_seizur = X_data[0]

    c_sample = np.array([amplitude, frequency]).flatten()

    energy_value_stimul = model.predict([energy_value_seizur.reshape(1, -1),
c_sample.reshape(1, -1), energy_value_normal.reshape(1, -1)]).flatten()
```

```
    weights = []
    for f in range(6):
        weights.append(abs(energy_value_seizur[f]-energy_value_normal[f]))
    weights = normalize_to_range(weights)
    cost_value = 0
    for f in range(6):
        cost_value += (weights[f])*(abs(energy_value_stimul[f]-energy_value_normal[f]))
    return cost_value


def MPC(initial_point, threshold, max_iterations=100, initial_std_amplitude=0.5,
initial_std_frequency=20, range_expansion_factor=0.1, no_progress_limit=5):
    current_point = initial_point
    current_cost = cost_function(*current_point)
    iteration = 0
    std_amplitude = initial_std_amplitude
    std_frequency = initial_std_frequency


    # Store points and costs for tracking
    points = [current_point]
    costs = [current_cost]
```

```
    no_progress_count = 0
    print(f"Iteration: {iteration}, Point: {current_point}, Cost: {current_cost}")
    while current_cost > threshold and iteration < max_iterations:
        best_cost = current_cost
        best_point = current_point
        # Generate 100 random points within the specified ranges
        for _ in range(10):
            candidate_amplitude = np.clip(np.random.normal(current_point[0], std_amplitude),
0.1, 5)
            candidate_frequency = np.clip(np.random.normal(current_point[1], std_frequency),
5, 200)
            candidate_point = np.array([candidate_amplitude, candidate_frequency])
            candidate_cost = cost_function(*candidate_point)
            if candidate_cost < best_cost:
                best_cost = candidate_cost
                best_point = candidate_point
        # Update the current point to the best candidate found
        if best_cost < current_cost:
            current_cost = best_cost
            current_point = best_point
```

```
            no_progress_count = 0  # Reset progress counter
            std_amplitude = initial_std_amplitude
            std_frequency = initial_std_frequency
        else:
            # If no better point is found, increase the standard deviation for a wider search
            no_progress_count += 1
            if no_progress_count >= no_progress_limit:
                std_amplitude += range_expansion_factor
                std_frequency += range_expansion_factor*3
                no_progress_count = 0  # Reset progress counter
        points.append(current_point)
        costs.append(current_cost)
        iteration += 1
        print(f"Iteration: {iteration}, Point: {current_point}, Cost: {current_cost}")
        # Early termination if cost function falls below the threshold
        if current_cost < threshold:
            break

    return points, costs
```

```
# Example usage

initial_point = np.array([0.5, 10.0])  # Initial guess for amplitude and frequency

threshold = -5  # Example threshold based on non-seizure episodes

points, costs = MPC(initial_point, threshold)

points1, costs1 = MPC (initial_point, threshold)

points2, costs2 = MPC(initial_point, threshold)

# Extract x, y coordinates from points

amplitudes, frequencies = zip(*points)

# Extract x, y coordinates from points

amplitudes, frequencies = zip(*points)

amplitudes1, frequencies1 = zip(*points1)

amplitudes2, frequencies2 = zip(*points2)
```

Appendix D VHDL Code for Patient-Optimized

Adaptive Predictive Controller

D.1 Fixed Point Package

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.MATH_REAL.ALL;

package fixed_point_pkg is
    subtype fix_point is signed(11 downto 0);
    constant Q: integer := 6;
    type fix_point_vector is array (integer range <>) of fix_point;
    type fix_point_matrix is array (integer range <>, integer range <>) of fix_point;

    function to_fixed_point (x: real) return fix_point;
    function to_real (x: fix_point) return integer;
    function add_fp (a, b: fix_point) return fix_point;
    function sub_fp (a, b: fix_point) return fix_point;
    function mul_fp (a, b: fix_point) return fix_point;
    function div_fp (a, b: fix_point) return fix_point;
    function square_fp (a: fix_point) return fix_point;
    function accumulate_energy (a, acc: fix_point) return fix_point;
    function abs_fp (a: fix_point) return fix_point;

    -- Declare the random_fp function as impure
    impure function random_fp return fix_point;

    -- Shared variables for seeds
    -- Shared variable and constants
shared variable seed1 : fix_point := to_fixed_point(31.0);
    constant a: fix_point := to_fixed_point(61.0);
    constant c: fix_point := to_fixed_point(57.0);
    constant m: fix_point := to_fixed_point(50.0);  -- Multiplier for the random generator
end package fixed_point_pkg;
```

```
package body fixed_point_pkg is
    function to_fixed_point (x: real) return fix_point is
    begin
        return to_signed(integer(x * (2.0**Q)), 12);
    end function;

    function to_real (x: fix_point) return integer is
    begin
        return to_integer(x);
    end function;

    function add_fp (a, b: fix_point) return fix_point is
    begin
        return a + b;
    end function;

    function sub_fp (a, b: fix_point) return fix_point is
    begin
        return a - b;
    end function;

    function mul_fp (a, b: fix_point) return fix_point is
        variable temp: signed (23 downto 0);
    begin
        temp := a * b;
        return resize(temp(Q+12 downto Q), 12);
    end function;

    function div_fp (a, b: fix_point) return fix_point is
        variable temp: signed (23 downto 0);
    begin
        temp := resize(a, 24) sll Q;
        return resize(temp / b, 12);
    end function;

    function square_fp (a: fix_point) return fix_point is
    begin
        return mul_fp(a, a);
    end function;

    function accumulate_energy (a, acc: fix_point) return fix_point is
```

```
        variable energy: fix_point;
    begin
        energy := add_fp(acc, square_fp(a));
        return energy;
    end function;

    function abs_fp (a: fix_point) return fix_point is
    begin
        if a < 0 then
            return -a;
        else
            return a;
        end if;
    end function;

    -- Convert the seeds to integers for proper bitwise operations
    impure function random_fp return fix_point is
        variable rand_fp: fix_point;
        variable seed1_int: fix_point;
        variable seed1_slv: fix_point;
        variable max_fp: fix_point;
    begin
        seed1_int := div_fp(mul_fp(a, seed1), m);
        seed1_int(5 downto 0):= (others => '0');
        seed1_slv := sub_fp(mul_fp(a, seed1), seed1_int);
        seed1 := seed1_slv;
        rand_fp := div_fp(seed1_slv,m);
    return rand_fp;
end function random_fp;
```

D.2 Deserializer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.ALL;

entity de_serializer is
  Port (clk:       in std_logic;
        reset:     in std_logic;
        data_input: in std_logic;
```

```
        clk_data: out std_logic;
        reset_out: out std_logic;
        sample: out fix_point;
        label_SN: out std_logic
         );
end de_serializer;

architecture Behavioral of de_serializer is
constant M : integer := 1023;
signal counter: integer range 0 to 13:= 0;
signal counter_s: integer range 0 to M:= 0;
signal temp_sample: fix_point:= (others => '0');
signal temp_label_SN: std_logic := '0';
signal temp_label: std_logic;
signal flag: std_logic:='0';
begin
process(clk,reset)
   begin
      if reset = '1' then
         reset_out <= '1';
         counter <= 0;
         counter_s <= 0;
         clk_data <= '0';
         sample <= temp_sample;
         temp_label <= '0';
         flag <= '0';
      elsif rising_edge(clk) then
         reset_out <= '0';
            if counter < 12 then
               clk_data <= '0';
               temp_sample(counter) <= data_input;
               counter <= counter+1;
            elsif counter = 12 then
               counter <= 0;
               counter_s<= counter_s+1;
               clk_data <= '1';
               sample <= temp_sample;
               if counter_s = M  then
                  counter_s <= 0;
                  temp_label <= '1';
```

```
                end if;
            end if;
        end if;
        label_SN <= temp_label;
    end process;
end Behavioral;
```

D.3 FIR BPF filter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.fixed_point_pkg.all;

entity fir_filter_1 is
    Port (
        clk          : in  std_logic;
        reset        : in  std_logic;
        input_data   : in  fix_point; -- Vector of input data
        energy       : out fix_point;              -- 32-bit energy output
        done_out     : out std_logic
    );
end fir_filter_1;

architecture Behavioral of fir_filter_1 is

    -- FIR filter order
    constant N : integer := 49;
    constant M : integer := 1023;
    -- FIR filter coefficients
    constant coeffs : fix_point_vector(0 to N) := (
    to_fixed_point(1.921904850969538905e-02),
    to_fixed_point(1.937326762542848635e-02),
    to_fixed_point(1.952172619619054611e-02),
    to_fixed_point(1.966434119849053488e-02),
    to_fixed_point(1.980103280895398332e-02),
    to_fixed_point(1.993172445772023460e-02),
    to_fixed_point(2.005634287968641877e-02),
```

to_fixed_point(2.017481816356075930e-02),
to_fixed_point(2.028708379868943840e-02),
to_fixed_point(2.039307671962273250e-02),
to_fixed_point(2.049273734838797517e-02),
to_fixed_point(2.058600963443837917e-02),
to_fixed_point(2.067284109224851521e-02),
to_fixed_point(2.075318283652906326e-02),
to_fixed_point(2.082698961503494720e-02),
to_fixed_point(2.089421983894303517e-02),
to_fixed_point(2.095483561077698259e-02),
to_fixed_point(2.100880274985900825e-02),
to_fixed_point(2.105609081526980952e-02),
to_fixed_point(2.109667312629991479e-02),
to_fixed_point(2.113052678037747809e-02),
to_fixed_point(2.115763266845940849e-02),
to_fixed_point(2.117797548787455145e-02),
to_fixed_point(2.119154375260952003e-02),
to_fixed_point(2.119832980102970624e-02),
to_fixed_point(2.119832980102970624e-02),
to_fixed_point(2.119154375260952003e-02),
to_fixed_point(2.117797548787455145e-02),
to_fixed_point(2.115763266845940849e-02),
to_fixed_point(2.113052678037747809e-02),
to_fixed_point(2.109667312629991479e-02),
to_fixed_point(2.105609081526980952e-02),
to_fixed_point(2.100880274985900825e-02),
to_fixed_point(2.095483561077698259e-02),
to_fixed_point(2.089421983894303517e-02),
to_fixed_point(2.082698961503494720e-02),
to_fixed_point(2.075318283652906326e-02),
to_fixed_point(2.067284109224851521e-02),
to_fixed_point(2.058600963443837917e-02),
to_fixed_point(2.049273734838797517e-02),
to_fixed_point(2.039307671962273250e-02),
to_fixed_point(2.028708379868943840e-02),
to_fixed_point(2.017481816356075930e-02),
to_fixed_point(2.005634287968641877e-02),
to_fixed_point(1.993172445772023460e-02),
to_fixed_point(1.980103280895398332e-02),
to_fixed_point(1.966434119849053488e-02),
to_fixed_point(1.952172619619054611e-02),

```
    to_fixed_point(1.937326762542848635e-02),
    to_fixed_point(1.921904850969538905e-02)
);

signal counter : integer range 0 to M := 0;
signal flag : std_logic := '0';
signal flag1 : std_logic := '0';

signal yf: fix_point_matrix(0 to M,0 to N):= (others => (others => (others => '0')));
signal input: fix_point_vector(0 to M):= (others => (others => '0'));

begin
    process(clk,reset)
    variable sum: fix_point;
    variable sum1: fix_point;
    begin
        if rising_edge(clk) then
            if reset = '1' then
                counter <= 0;
                flag <= '0';
                flag1<= '0';
                energy <= (others => '0');
                done_out <= '0';
                yf <= (others => (others => (others => '0')));
                input <= (others => (others => '0'));
            elsif flag = '0' then
                if counter<=M then
                    counter <= counter + 1;
                    input(counter)<= input_data;
                else
                    flag <= '1';
                end if;
            elsif flag1 = '0' then
                yf(0,0)<= mul_fp(coeffs(0),input(0));
                for i in 0 to M loop
                    if i<N then
                        for j in 0 to i loop
                            yf(i,j)<= mul_fp(coeffs(j),input(i-j));
                        end loop;
                    else
```

```
                for j in 0 to N loop
                    yf(i,j)<= mul_fp(coeffs(j),input(i-j));
                end loop;
            end if;
        end loop;
        flag1 <= '1';
    else
        sum1 := to_fixed_point(0.0);
        for i in 0 to M loop
            sum := to_fixed_point(0.0);
            for j in 0 to N loop
                sum := add_fp(sum, yf(i,j));
            end loop;
            sum1 := add_fp(sum1, square_fp(sum));
        end loop;
        energy <= sum1;
        done_out <= '1';
    end if;
  end if;
 end process;
end Behavioral;
```

D.4 Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.all;

entity FIR_filters is
   Port (
        clk_data    : in  std_logic;
        clk         : in  std_logic;
        reset       : in  std_logic;
        input_data  : in  fix_point; -- Vector of input data
        label_SN    : in  std_logic;          -- 32-bit energy output
        done        : out std_logic;
        wait_input  : out std_logic;
        S_E1        : out fix_point;
```

```
        S_E2        : out fix_point;
        S_E3        : out fix_point;
        S_E4        : out fix_point;
        S_E5        : out fix_point;
        S_E6        : out fix_point;
        N_E1        : out fix_point;
        N_E2        : out fix_point;
        N_E3        : out fix_point;
        N_E4        : out fix_point;
        N_E5        : out fix_point;
        N_E6        : out fix_point
    );
end FIR_filters;

architecture Behavioral of FIR_filters is
-- Component declaration for `fir_filter_1`
  component fir_filter_1
    Port (
      clk       : in  std_logic;
      reset     : in  std_logic;
      input_data : in  fix_point;
      energy    : out fix_point;
      done_out  : out std_logic
    );
  end component;
  component fir_filter_2
    Port (
      clk       : in  std_logic;
      reset     : in  std_logic;
      input_data : in  fix_point;
      energy    : out fix_point;
      done_out  : out std_logic
    );
  end component;
  component fir_filter_3
    Port (
      clk       : in  std_logic;
      reset     : in  std_logic;
      input_data : in  fix_point;
      energy    : out fix_point;
```

```
        done_out   : out std_logic
    );
  end component;
  component fir_filter_4
    Port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        input_data : in  fix_point;
        energy    : out fix_point;
        done_out   : out std_logic
    );
  end component;
  component fir_filter_5
    Port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        input_data : in  fix_point;
        energy    : out fix_point;
        done_out   : out std_logic
    );
  end component;
  component fir_filter_66
    Port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        input_data : in  fix_point;
        energy    : out fix_point;
        done_out   : out std_logic
    );
  end component;


constant M : integer := 1023;
signal reset_firs: std_logic:='1';
signal s_input: fix_point_vector(M downto 0):= (others => (others => '0'));
signal n_input: fix_point_vector(M downto 0):= (others => (others => '0'));
signal firs_input: fix_point := (others => '0');
signal flag : std_logic := '0';
signal flag_n: std_logic:= '0';
signal flag_s: std_logic:= '0';
signal flag_d: std_logic:= '0';
```

```
signal fir_done_1,fir_done_2,fir_done_3,fir_done_4,fir_done_5,fir_done_6 : std_logic;
signal energy1,energy2,energy3,energy4,energy5,energy6 : fix_point;
signal counter_s: integer range 0 to M+1 := 0;
signal counter_n: integer range 0 to M+1 := 0;
signal counter_SS: integer range 0 to M+1 := 0;
signal counter_NN: integer range 0 to M+1 := 0;

begin

process(clk_data,reset)
begin
   if reset = '1' then
      counter_n <= 0;
      counter_s<= 0;
      flag_n <= '0';
      flag_s <= '0';
   elsif rising_edge(clk_data) then
       if counter_n<=M then
         n_input(counter_n)  <= input_data;
         counter_n <= counter_n+1;
         flag_n <= '0';
       else
         flag_n <= '1';
         if counter_s<=M then
            s_input(counter_s)  <= input_data;
            counter_s <= counter_s+1;
            flag_s <= '0';
         else
            flag_s <= '1';
         end if;
      end if;
  end if;
end process;

   process(clk,reset)
   begin
      if reset = '1' then
         S_E1 <= (others => '0');
         S_E2 <= (others => '0');
         S_E3 <= (others => '0');
```

```
        S_E4 <= (others => '0');
        S_E5 <= (others => '0');
        S_E6 <= (others => '0');
        N_E1 <= (others => '0');
        N_E2 <= (others => '0');
        N_E3 <= (others => '0');
        N_E4 <= (others => '0');
        N_E5 <= (others => '0');
        N_E6 <= (others => '0');
        counter_NN <= 0;
        reset_firs <= '1';
        flag      <= '0';
        flag_d    <= '0';
      elsif rising_edge(clk) then
        if flag_n = '1' and flag= '0' then
          reset_firs <= '0';
          if counter_NN <= M then
            s_input(counter_NN)  <= input_data;
            counter_NN <= counter_NN + 1;
            flag      <= '0';
          elsif fir_done_1 = '1' and fir_done_2 = '1' and fir_done_3 = '1' and fir_done_4 = '1'
and fir_done_5 = '1' and fir_done_6 = '1' then
            N_E1 <= energy1;
            N_E2 <= energy2;
            N_E3 <= energy3;
            N_E4 <= energy4;
            N_E5 <= energy5;
            N_E6 <= energy6;
            reset_firs <= '1';
            flag <= '1';       --normal energy is calculated
          end if;
        end if;
      end if;
      if reset = '1' then
        S_E1 <= (others => '0');
        S_E2 <= (others => '0');
        S_E3 <= (others => '0');
        S_E4 <= (others => '0');
        S_E5 <= (others => '0');
        S_E6 <= (others => '0');
        N_E1 <= (others => '0');
```

```
            N_E2 <= (others => '0');
            N_E3 <= (others => '0');
            N_E4 <= (others => '0');
            N_E5 <= (others => '0');
            N_E6 <= (others => '0');
            counter_NN <= 0;
            counter_SS <= 0;
            reset_firs <= '1';
            flag_d <= '0';
        elsif rising_edge(clk) then
            if flag_s = '1' and flag = '1' then
                reset_firs <= '0';
                if counter_SS <= M then
                    s_input(counter_SS)  <= input_data;
                    counter_SS <= counter_SS + 1;
                    flag_d <= '0';
                elsif fir_done_1 = '1' and fir_done_2 = '1' and fir_done_3 = '1' and fir_done_4 = '1'
and fir_done_5 = '1' and fir_done_6 = '1' then
                    S_E1 <= energy1;
                    S_E2 <= energy2;
                    S_E3 <= energy3;
                    S_E4 <= energy4;
                    S_E5 <= energy5;
                    S_E6 <= energy6;
                    reset_firs <= '1';
                    flag_d <= '1';
                end if;
            end if;
        end if;
        done <= not(flag_d);
    end process;

U1: fir_filter_1
        Port map (
            clk => clk,
            reset => reset_firs,
            input_data => firs_input,
            energy => energy1,
            done_out => fir_done_1
        );
U2: fir_filter_2
```

```
        Port map (
            clk => clk,
            reset => reset_firs,
            input_data => firs_input,
            energy => energy2,
            done_out => fir_done_2
        );
U3: fir_filter_3
        Port map (
            clk => clk,
            reset => reset_firs,
            input_data => firs_input,
            energy => energy3,
            done_out => fir_done_3
        );
U4: fir_filter_4
        Port map (
            clk => clk,
            reset => reset_firs,
            input_data => firs_input,
            energy => energy4,
            done_out => fir_done_4
        );
U5: fir_filter_5
        Port map (
            clk => clk,
            reset => reset_firs,
            input_data => firs_input,
            energy => energy5,
            done_out => fir_done_5
        );
U6: fir_filter_66
        Port map (
            clk => clk,
            reset => reset_firs,
            input_data => firs_input,
            energy => energy6,
            done_out => fir_done_6
        );

end Behavioral;
```

D.5 One Layer of the Neural Network

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.ALL;

entity encoder_L1 is
    port (
        clk: in STD_LOGIC;
        reset : in STD_LOGIC;
        input_data: in fix_point_vector(0 to 5);
        output_data: out fix_point_vector(0 to 19);
        done: out STD_LOGIC
    );
end encoder_L1;

architecture Behavioral of encoder_L1 is

constant N : integer := 6-1;
constant M : integer := 20-1;

--signal input_data: fix_point_vector(0 to N):=(others => (others => '0'));
--signal input_data: fix_point_vector(0 to N):=
(to_fixed_point(2.0),to_fixed_point(1.0),to_fixed_point(4.0),to_fixed_point(1.0),to_fixed_poi
nt(6.0),to_fixed_point(1.0));
--signal weights: fix_point_matrix(0 to N, 0 to M):= (others =>(others => (others => '0')));
signal weights: fix_point_matrix(0 to 5, 0 to 19) := (
    (to_fixed_point(-8.312019705772399902e-02), to_fixed_point(1.711004376411437988e-
01), to_fixed_point(-3.426226377487182617e-01), to_fixed_point(9.654361009597778320e-
02), to_fixed_point(-3.502904176712036133e-01), to_fixed_point(1.667172759771347046e-
01), to_fixed_point(3.306994438171386719e-01), to_fixed_point(-4.225176870822906494e-
01), to_fixed_point(-1.311010271310806274e-01), to_fixed_point(4.000863432884216309e-
01), to_fixed_point(4.168120026588439941e-01), to_fixed_point(2.958077788352966309e-
01), to_fixed_point(1.066751182079315186e-01), to_fixed_point(1.774274706840515137e-
01), to_fixed_point(4.972648620605468750e-02), to_fixed_point(1.739683747291564941e-
01), to_fixed_point(2.869607210159301758e-01), to_fixed_point(-3.717790842056274414e-
01), to_fixed_point(1.991309821605682373e-01), to_fixed_point(9.511828422546386719e-
03)),
```

(to_fixed_point(3.803762197494506836e-01), to_fixed_point(4.596654176712036133e-01), to_fixed_point(2.397148609161376953e-01), to_fixed_point(-4.699311852455139160e-01), to_fixed_point(4.604554176330566406e-01), to_fixed_point(5.329342484474182129e-01), to_fixed_point(4.028192162513732910e-01), to_fixed_point(6.184291839599609375e-02), to_fixed_point(-3.865794837474822998e-01), to_fixed_point(3.630170822143554688e-01), to_fixed_point(-1.841915249824523926e-01), to_fixed_point(4.643744826316833496e-01), to_fixed_point(-4.343289434909820557e-01), to_fixed_point(1.560484431684017181e-02), to_fixed_point(-1.546432673931121826e-01), to_fixed_point(-3.371205925941467285e-01), to_fixed_point(8.390539884567260742e-02), to_fixed_point(2.045756727457046509e-01), to_fixed_point(1.077664941549301147e-01), to_fixed_point(1.3017141819000244140e-01)),
(to_fixed_point(-5.016425251960754395e-02), to_fixed_point(-2.974602580070495605e-01), to_fixed_point(-3.531485795974731445e-02), to_fixed_point(-2.604731917381286621e-01), to_fixed_point(2.549418509006500244e-01), to_fixed_point(1.868201345205307007e-01), to_fixed_point(-3.531579971313476562e-01), to_fixed_point(-3.591267466545104980e-01), to_fixed_point(-6.885616481304168701e-02), to_fixed_point(7.399356365203857422e-02), to_fixed_point(1.126314401626586914e-01), to_fixed_point(-3.772822618484497070e-01), to_fixed_point(-4.345020949840545654e-01), to_fixed_point(1.020178385078907013e-02), to_fixed_point(-1.264551877975463867e-01), to_fixed_point(-2.867059707641601562e-01), to_fixed_point(-3.690804541110992432e-01), to_fixed_point(-3.621934652328491211e-01), to_fixed_point(4.069026112556457520e-01), to_fixed_point(-2.338371723890304565e-01)),
(to_fixed_point(-8.321902155876159668e-02), to_fixed_point(-3.606201410293579102e-01), to_fixed_point(8.908754587173461914e-02), to_fixed_point(3.061661124229431152e-01), to_fixed_point(-1.117008104920387268e-01), to_fixed_point(1.640984714031219482e-01), to_fixed_point(-2.222013473510742188e-02), to_fixed_point(-3.932097554206848145e-02), to_fixed_point(2.542961537837982178e-01), to_fixed_point(-3.089292049407958984e-01), to_fixed_point(8.418130874633789062e-02), to_fixed_point(3.735889196395874023e-01), to_fixed_point(2.551046311855316162e-01), to_fixed_point(2.975482344627380371e-01), to_fixed_point(-2.107807397842407227e-01), to_fixed_point(-2.223390042781829834e-01), to_fixed_point(-3.595304787158966064e-01), to_fixed_point(4.099327325820922852e-01), to_fixed_point(4.498593509197235107e-01), to_fixed_point(-2.529905438423156738e-01)),
(to_fixed_point(-3.798836171627044678e-01), to_fixed_point(-2.407480627298355103e-01), to_fixed_point(-4.724124073982238770e-02), to_fixed_point(-2.248026132583618164e-01), to_fixed_point(3.696001172065734863e-01), to_fixed_point(-2.642624676227569580e-01), to_fixed_point(-3.850382566452026367e-01), to_fixed_point(-3.878470659255981445e-01), to_fixed_point(4.062012135982513428e-01), to_fixed_point(-3.581169247627258301e-02), to_fixed_point(-4.305189251899719238e-01), to_fixed_point(1.802602410316467285e-01), to_fixed_point(4.534354209899902344e-01), to_fixed_point(8.201515674591064453e-02), to_fixed_point(-4.196074008941650391e-01), to_fixed_point(5.559134483337402344e-02), to_fixed_point(4.239292740821838379e-01), to_fixed_point(-4.412021040916442871e-

```
01), to_fixed_point(-3.660696744918823242e-01), to_fixed_point(-1.455487012863159180e-
01)),
   (to_fixed_point(1.878480911254882812e-01), to_fixed_point(-4.353991150856018066e-
02), to_fixed_point(-4.073848426342010498e-01), to_fixed_point(-1.817159056663513184e-
01), to_fixed_point(5.078807473182678223e-02), to_fixed_point(5.302458629012107849e-
02), to_fixed_point(-1.049181520938873291e-01), to_fixed_point(-1.911445856094360352e-
01), to_fixed_point(4.671766161918640137e-01), to_fixed_point(-2.075964808464050293e-
01), to_fixed_point(-3.840383887290954590e-01), to_fixed_point(-2.552140951156616211e-
01), to_fixed_point(1.405642777681350708e-01), to_fixed_point(-2.970128133893013000e-
02), to_fixed_point(-1.331250071525573730e-01), to_fixed_point(-2.573074102401733398e-
01), to_fixed_point(-4.294747412204742432e-01), to_fixed_point(3.098889291286468506e-
01), to_fixed_point(-3.107199668884277344e-01), to_fixed_point(5.513036251068115234e-
02))
);

--signal biases: fix_point_vector(0 to M):=(others => (others => '0'));
signal biases: fix_point_vector(0 to 19) := (
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(1.201200038194656372e-01),
   to_fixed_point(1.859568506479263306e-01),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(1.499918401241302490e-01),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(-2.185068093240261078e-02),
   to_fixed_point(1.630085408687591553e-01),
   to_fixed_point(-1.164130941033363342e-01),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(0.00000000000000000e+00),
   to_fixed_point(-6.186761893332004547e-04),
   to_fixed_point(1.549838930368423462e-01),
   to_fixed_point(0.00000000000000000e+00)
);
```

```
signal temp: fix_point_matrix(0 to N, 0 to M):= (others =>(others => (others => '0')));
--signal output_data: fix_point_vector(0 to M):=(others => (others => '0'));
signal flag : std_logic := '0';
signal flag1 : std_logic := '0';

begin
process(clk,reset)
variable sum: fix_point;
   begin
      if reset = '1' then
         flag <= '0';
         flag1 <= '0';
         done <= '1';
         temp <= (others =>(others => (others => '0')));
         output_data <= (others => (others => '0'));
      elsif rising_edge(clk) then
         if flag = '0' and flag1 = '0' then
            for i in 0 to M loop
               for j in 0 to N loop
                  temp(j,i) <= mul_fp(input_data(j),weights(j,i));
               end loop;
            end loop;
            flag <= '1';
            done <= '1';
         elsif flag1 = '0' then
            --for i in 0 to M loop
            --    output_data(i) <=
add_fp(temp(0,i),(add_fp(temp(1,i),(add_fp(temp(2,i),(add_fp(temp(3,i),(add_fp(temp(4,i),add
_fp(temp(5,i),biases(i)))))))))));
            --end loop;
            for i in 0 to M loop
               sum:= biases(i);
               for j in 0 to N loop
                  sum := add_fp(sum, temp(j, i));
               end loop;
               if sum(11) = '0' then
                  output_data(i) <= sum;
               else
                  output_data(i) <= (others => '0');
```

```
                end if;
            end loop;

            done <= '0';
            flag1 <= '1';
        end if;
    end if;
  end process;
end Behavioral;
```

D.6 Encoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.fixed_point_pkg.ALL;

entity Encoder is
  Port (
    clk: in STD_LOGIC;
    reset: in STD_LOGIC;
    input_data: in fix_point_vector(0 to 5);
    output_data: out fix_point_vector(0 to 19);
    done: out STD_LOGIC
  );
end Encoder;

architecture Behavioral of Encoder is

  -- Component declaration for encoder_L1
  component encoder_L1 is
    Port (
      clk : in STD_LOGIC;
      reset : in STD_LOGIC;
      input_data : in fix_point_vector(0 to 5);
      output_data : out fix_point_vector(0 to 19);
      done : out STD_LOGIC
```

```
    );
end component;



-- Component declaration for encoder_L2
component encoder_L2 is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        input_data : in fix_point_vector(0 to 19);
        output_data : out fix_point_vector(0 to 19);
        done : out STD_LOGIC
    );
end component;

-- Component declaration for encoder_L3
component encoder_L3 is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        input_data : in fix_point_vector(0 to 19);
        output_data : out fix_point_vector(0 to 19);
        done : out STD_LOGIC
    );
end component;

-- Component declaration for encoder_L4
component encoder_L4 is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        input_data : in fix_point_vector(0 to 19);
        output_data : out fix_point_vector(0 to 19);
        done : out STD_LOGIC
    );
end component;
component encoder_L5 is
    Port (
        clk : in STD_LOGIC;
```

```
        reset : in STD_LOGIC;
        input_data : in fix_point_vector(0 to 19);
        output_data : out fix_point_vector(0 to 19);
        done : out STD_LOGIC
    );
end component;

-- Signal declarations to connect internal blocks
signal output_data_0, output_data_1, output_data_2,output_data_3 : fix_point_vector(0 to
19);
signal done_0, done_1, done_2, done_3 : STD_LOGIC;



begin



-- Instantiate the first encoder_L1 block
encoder1: encoder_L1
    port map (
        clk => clk,
        reset => reset,
        input_data => input_data,
        output_data => output_data_0,
        done => done_0
    );

-- Instantiate the second encoder_L2 block
encoder2: encoder_L2
    port map (
        clk => clk,
        reset => done_0,
        input_data => output_data_0,
        output_data => output_data_1,
        done => done_1
    );

-- Instantiate the third encoder_L3 block
encoder3: encoder_L3
```

```
        port map (
            clk => clk,
            reset => done_1,
            input_data => output_data_1,
            output_data => output_data_2,
            done => done_2
        );

    -- Instantiate the fourth encoder_L4 block
    encoder4: encoder_L4
        port map (
            clk => clk,
            reset => done_2,
            input_data => output_data_2,
            output_data => output_data_3,
            done => done_3
        );
    encoder5: encoder_L5
        port map (
            clk => clk,
            reset => done_3,
            input_data => output_data_3,
            output_data => output_data,
            done => done
        );
end Behavioral;
```

D.7 Koopman Block

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.ALL;

entity Koopman is
    Port (
        clk: in STD_LOGIC;
        reset_A: in STD_LOGIC;
        reset_B: in STD_LOGIC;
```

```
        input_data: in fix_point_vector(0 to 19);
        control_signal: in fix_point_vector(0 to 1);
        output_data: out fix_point_vector(0 to 19);
        done: out STD_LOGIC
    );
end Koopman;

architecture Behavioral of Koopman is
constant M : integer := 20-1;
signal out_A, out_B: fix_point_vector(0 to 19);
signal done_A, done_B: std_logic;
begin

A: entity work.Koopman_A
        port map (
            clk => clk,
            reset => reset_A,
            input_data => input_data,
            output_data => out_A,
            done => done_A
        );
B: entity work.Koopman_B
        port map (
            clk => clk,
            reset => reset_B,
            input_data => control_signal,
            output_data => out_B,
            done => done_B
        );

process(clk)
begin
    if rising_edge(clk) then
        if done_A = '0' and done_B = '0' then
            for i in 0 to M loop
                output_data(i) <= add_fp(out_A(i), out_B(i));
            end loop;
            done <= '0';
        else
            done <= '1';
```

```
            output_data <= (others => (others => '0'));
        end if;
    end if;
end process;

end Behavioral;
```

D.8 Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.fixed_point_pkg.ALL;

entity Decoder is
    Port (
        clk: in STD_LOGIC;
        reset: in STD_LOGIC;
        input_data: in fix_point_vector(0 to 19);
        output_data: out fix_point_vector(0 to 5);
        done: out STD_LOGIC
    );
end Decoder;

architecture Behavioral of Decoder is

    -- Signal declarations to connect internal blocks
    signal output_data_0, output_data_1, output_data_2, output_data_3  : fix_point_vector(0
to 19);
    signal done_0, done_1, done_2, done_3 : STD_LOGIC;

begin

    -- Instantiate the first encoder_L1 block
    decoder1: entity work.decoder_L1
        port map (
            clk => clk,
```

```
        reset => reset,
        input_data => input_data,
        output_data => output_data_0,
        done => done_0
    );

-- Instantiate the second encoder_L1 block
decoder2: entity work.decoder_L2
    port map (
        clk => clk,
        reset => done_0,
        input_data => output_data_0,
        output_data => output_data_1,
        done => done_1
    );

-- Instantiate the third encoder_L1 block
decoder3: entity work.decoder_L3
    port map (
        clk => clk,
        reset => done_1,
        input_data => output_data_1,
        output_data => output_data_2,
        done => done_2
    );

-- Instantiate the fourth encoder_L1 block
decoder4: entity work.decoder_L4
    port map (
        clk => clk,
        reset => done_2,
        input_data => output_data_2,
        output_data => output_data_3,
        done => done_3
    );
decoder5: entity work.decoder_L5
    port map (
        clk => clk,
        reset => done_2,
        input_data => output_data_3,
```

```
        output_data => output_data,
        done => done
    );

end Behavioral;
```

D.9 Cost Function Calculator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.fixed_point_pkg.all;

entity cost_function is
    port(
        clk: in std_logic;
        reset:  in std_logic;
        normal: in fix_point_vector(0 to 5);
        seizure: in fix_point_vector(0 to 5);
        stimula: in fix_point_vector(0 to 5);
        cost_value: out fix_point;
        done: out std_logic);
end cost_function;

architecture Behavioral of cost_function is
signal weights: fix_point_vector(0 to 5):= (others => (others => '0'));
signal cost : fix_point_vector(0 to 5):= (others => (others => '0'));
signal temp: fix_point;
signal max: fix_point:= (others => '0');
signal min: fix_point:= (others => '0');
signal flag1: std_logic:='0';
signal flag2: std_logic:='0';
signal flag3: std_logic:='0';
begin

process(clk)
    begin
        if reset = '1' then
```

```
            done <= '0';
            cost_value <= to_fixed_point(0.0);
            weights <= (others => (others => '0'));
            cost    <= (others => (others => '0'));
            max <= (others => '0');
            flag1<= '0';
            flag2<= '0';
            flag3<= '0';
        elsif flag1 = '0' and flag2 = '0'then
            cost_value<=to_fixed_point(0.0);
            for i in 0 to 5 loop
                weights(i) <= abs_fp(sub_fp(normal(i),seizure(i)));
                if max<abs_fp(sub_fp(normal(i),seizure(i))) then
                    max<=abs_fp(sub_fp(normal(i),seizure(i)));
                end if;
                if min>abs_fp(sub_fp(normal(i),seizure(i))) then
                    min<=abs_fp(sub_fp(normal(i),seizure(i)));
                end if;
            end loop;
            flag1 <= '1';
        elsif flag2 = '0' then
            flag2 <= '1';
        elsif flag3='0' then
            for i in 0 to 5 loop
                cost(i)<= mul_fp(weights(i),abs_fp(sub_fp(normal(i),stimula(i))));
            end loop;
            flag3<='1';
        else
            cost_value<=
add_fp(cost(0),(add_fp(cost(1),(add_fp(cost(2),(add_fp(cost(3),(add_fp(cost(4),cost(5))))))))));
            done <= '1';
        end if;
    end process;

end Behavioral;
```

D.10 MPC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.ALL;

entity MPC is
  Port (
        clk:       in std_logic;
        reset:     in std_logic;
        label_SN:  in  std_logic;
        new_control: in std_logic;
        cost_value: in fix_point;
        control_signal_o: out fix_point_vector(0 to 1);
        control_signal_f: out fix_point_vector(0 to 1);
        done_c          : out std_logic;
        done          : out std_logic
        );
end MPC;

architecture Behavioral of MPC is
    type state_type is (State0, State1, State2, State3, State4,state5);
    signal current_state, next_state : state_type;
    signal initial_point: fix_point_vector(0 to 1):= (to_fixed_point(0.05), to_fixed_point(10.0));
    signal save_initial_point: fix_point_vector(0 to 1):= (to_fixed_point(0.05),
to_fixed_point(10.0));
    signal random_point:  fix_point_vector(0 to 1):= (others => (others => '0'));
    signal cost_value_min: fix_point := to_fixed_point(10000.0);
    signal control_temp : fix_point_vector(0 to 1):= (to_fixed_point(0.0), to_fixed_point(0.0));
    signal std_dev_fixed_amp: fix_point := to_fixed_point(0.05);  -- Convert std_dev to fixed-
point
    signal std_dev_fixed_freq: fix_point := to_fixed_point(2.0);
    signal counter1 : integer range 0 to 10 := 0;
    signal counter2 : integer range 0 to 10 := 0;
    signal seed1: fix_point := to_fixed_point(0.52);
    signal seed1_int: fix_point := to_fixed_point(0.52);
    constant a: fix_point := to_fixed_point(61.0);
```

```
    constant c: fix_point := to_fixed_point(57.0);
    constant m: fix_point := to_fixed_point(50.0);  -- Multiplier for the random generator

begin
    -- State register
    process (clk, reset)
    begin
        if reset = '1' or label_SN = '0' then
            current_state <= State0;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    -- Next state logic
    process (current_state,clk)
    begin
        case current_state is
            when State0 =>
                control_signal_o <= (others => (others => '0'));
                control_signal_f <= (others => (others => '0'));
                done_c          <= '0';
                done            <= '0';
                counter1        <= 0;
                next_state <= State1;

            when State1 =>
                done_c          <= '0';
                done            <= '0';
                control_temp <= initial_point;
                if new_control ='1' then
                    next_state <= State2;
                    done_c <= '0';
                end if;

            when State2 =>
                done_c          <= '1';
                if to_real(cost_value_min)>to_real(cost_value) then
                    save_initial_point <= control_temp;
                    end if;
```

```
            seed1_int <= div_fp(mul_fp(a, seed1), m);
            seed1_int(5 downto 0)<= (others => '0');
            seed1 <= sub_fp(mul_fp(a, seed1), seed1_int);
            control_temp(0) <=
add_fp(initial_point(0),mul_fp(div_fp(seed1,m),std_dev_fixed_amp));
            control_temp(1) <=
add_fp(initial_point(1),mul_fp(div_fp(seed1,m),std_dev_fixed_freq));
            counter1 <= counter1 + 1;
            next_state <= State3;
         when State3 =>
            done_c         <= '0';
            if new_control = '1' then
               if counter1 < 10 then
                  next_state <= State2;
               else
                  next_state <= State4;
               end if;
            end if;
         when State4 =>
            initial_point <= save_initial_point;
            counter2 <= counter2 + 1;
            counter1 <= 0;
            if counter2 < 10 then
               next_state <= State2;
            else
               control_signal_f <= save_initial_point;
               done             <= '1';
            end if;
         when others =>
            next_state <= State0;
      end case;
       control_signal_o <= control_temp;
   end process;

end Behavioral;
```

D.11 Adaptive Controller Top Level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.all;

entity Adaptive_controller_top_block is
  Port (
       Internal_clk   : in  std_logic;
       clk_input      : in std_logic;
       reset        : in  std_logic;
       input_data   : in  fix_point; -- Vector of input data
       label_SN     : in  std_logic;           -- 32-bit energy output
       control_signal_fs: out fix_point_vector(0 to 1);
       done_out : out std_logic
   );
end Adaptive_controller_top_block;



architecture Behavioral of Adaptive_controller_top_block is

component FIR_filters is
    Port (
          clk_data     : in  std_logic;
          clk          : in  std_logic;
          reset        : in  std_logic;
          input_data   : in  fix_point; -- Vector of input data
          label_SN     : in  std_logic;           -- 32-bit energy output
          done         : out std_logic;
          wait_input   : out std_logic;
          S_E1         : out fix_point;
          S_E2         : out fix_point;
          S_E3         : out fix_point;
          S_E4         : out fix_point;
          S_E5         : out fix_point;
          S_E6         : out fix_point;
```

```
            N_E1        : out fix_point;
            N_E2        : out fix_point;
            N_E3        : out fix_point;
            N_E4        : out fix_point;
            N_E5        : out fix_point;
            N_E6        : out fix_point

        );
end component;

component Encoder is
    Port (
        clk: in STD_LOGIC;
        reset: in STD_LOGIC;
        input_data: in fix_point_vector(0 to 5);
        output_data: out fix_point_vector(0 to 19);
        done: out STD_LOGIC
    );
end component;

component Koopman is
    Port (
        clk: in STD_LOGIC;
        reset_A: in STD_LOGIC;
        reset_B: in STD_LOGIC;
        input_data: in fix_point_vector(0 to 19);
        control_signal: in fix_point_vector(0 to 1);
        output_data: out fix_point_vector(0 to 19);
        done: out STD_LOGIC
    );
end component;

component Decoder is
    Port (
        clk: in STD_LOGIC;
        reset: in STD_LOGIC;
        input_data: in fix_point_vector(0 to 19);
        output_data: out fix_point_vector(0 to 5);
        done: out STD_LOGIC
```

```
   );
end component;

component cost_function is
   port(
        clk: in std_logic;
        reset:  in std_logic;
        normal: in fix_point_vector(0 to 5);
        seizure: in fix_point_vector(0 to 5);
        stimula: in fix_point_vector(0 to 5);
        cost_value: out fix_point;
        done: out std_logic);
end component;

component MPC is
  Port (
        clk:       in std_logic;
        reset:     in std_logic;
        label_SN:  in  std_logic;
        new_control: in std_logic;
        cost_value: in fix_point;
        control_signal_o: out fix_point_vector(0 to 1);
        control_signal_f: out fix_point_vector(0 to 1);
        done_c          : out std_logic;
        done          : out std_logic
        );
end component;

-- filters signals
signal reset_FIRs: std_logic;
signal r_done_FIRs: std_logic;
signal wait_input : std_logic;
signal S_E1         : fix_point := (others => '0');
signal S_E2         : fix_point := (others => '0');
signal S_E3         : fix_point := (others => '0');
signal S_E4         : fix_point := (others => '0');
signal S_E5         : fix_point := (others => '0');
signal S_E6         : fix_point := (others => '0');
signal N_E1         : fix_point := (others => '0');
signal N_E2         : fix_point := (others => '0');
```

```
signal N_E3        : fix_point := (others => '0');
signal N_E4        : fix_point := (others => '0');
signal N_E5        : fix_point := (others => '0');
signal N_E6        : fix_point := (others => '0');

--encoder signals
signal reset_encoder: std_logic;
signal current_data: fix_point_vector(0 to 5);
signal lifted_data: fix_point_vector(0 to 19);
signal r_done_encoder: std_logic;

-- koopman signals
signal reset_A,reset_B: std_logic;
signal control_signal: fix_point_vector(0 to 1);
signal next_lifted_data: fix_point_vector(0 to 19);
signal r_done_koopman: std_logic;

--decoder
signal reset_decoder, r_done_decoder: std_logic;
signal next_data: fix_point_vector(0 to 5);

-- cost function
signal reset_CF,done_CF: std_logic;
signal cost_value: fix_point;

--MPC
signal reset_MPC,new_control,done_c,done_MPC: std_logic;
signal control_signal_o: fix_point_vector(0 to 1);
signal control_signal_f: fix_point_vector(0 to 1);

--counter
signal horizon: integer range 0 to 1;
signal state  : integer range 0 to 10;
begin

u1: FIR_filters
   port map(
            clk_data  => clk_input,
            clk       => internal_clk,
            reset     => reset,
```

```
            input_data=> input_data,
            label_SN  => label_SN,
            done      => r_done_FIRs,
            wait_input=> wait_input,
            S_E1      => S_E1,
            S_E2      => S_E2,
            S_E3      => S_E3,
            S_E4      => S_E4,
            S_E5      => S_E5,
            N_E1      => N_E1,
            N_E2      => N_E2,
            N_E3      => N_E3,
            N_E4      => N_E4,
            N_E5      => N_E5
            );
u2: Encoder
   port map(
            clk       => internal_clk,
            reset     => r_done_FIRs,
            input_data(0)=> S_E1,
            input_data(1)=> S_E2,
            input_data(2)=> S_E3,
            input_data(3)=> S_E4,
            input_data(4)=> S_E5,
            input_data(5)=> S_E6,
            output_data=> lifted_data,
            done      => r_done_encoder
            );
u3: Koopman
   port map(
            clk       => internal_clk,
            reset_A   => r_done_encoder,
            reset_B   => done_c,
            input_data=> lifted_data,
            control_signal => control_signal_o,
            output_data=> next_lifted_data,
            done      => r_done_koopman
            );
u4: Decoder
   port map(
            clk       => internal_clk,
```

```
            reset     => r_done_koopman,
            input_data=> next_lifted_data,
            output_data=> next_data,
            done      => r_done_decoder
            );

u5: cost_function
   port map(
            clk       => internal_clk,
            reset     => r_done_decoder,
            normal(0) => N_E1,
            normal(1) => N_E2,
            normal(2) => N_E3,
            normal(3) => N_E4,
            normal(4) => N_E5,
            normal(5) => N_E6,
            seizure(0)=> S_E1,
            seizure(1)=> S_E2,
            seizure(2)=> S_E3,
            seizure(3)=> S_E4,
            seizure(4)=> S_E5,
            seizure(5)=> S_E6,
            stimula   => next_data,
            cost_value=> cost_value,
            done      => done_CF
            );

u6: MPC
   port map(
            clk       => internal_clk,
            reset     => reset,
            label_SN  => label_SN,
            new_control=> done_CF,
            cost_value=> cost_value,
            control_signal_o => control_signal_o,
            control_signal_f => control_signal_fs,
            done_c    => done_c,
            done      => done_out
            );

end Behavioral;
```

D.12 Serializer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.fixed_point_pkg.ALL;

entity serializer is
Port ( clk: in std_logic;
       control_signal_fs: in fix_point_vector(0 to 1);
       done_in : in std_logic;
       amp_serial: out std_logic;
       freq_serial: out std_logic;
       start: out std_logic;
       finish: out std_logic
   );
end serializer;

architecture Behavioral of serializer is
signal counter : integer range 0 to 12;
begin
process (clk)
   begin
      if done_in = '0' then
         amp_serial <= '0';
         freq_serial<= '0';
         start <= '0';
         finish <= '0';
         counter <= 0;
      else
         if counter < 11 then
            start <= '1';
            amp_serial <= control_signal_fs(0)(counter);
            freq_serial <= control_signal_fs(1)(counter);
            counter <= counter +1;
         else
            start <= '0';
            finish <= '1';
```

```
            end if;
        end if;
    end process;
end Behavioral;
```

D.13 Top Level System

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.fixed_point_pkg.all;

-- Top-level entity that connects de_serializer, Adaptive_controller_top_block, and serializer
entity Deep_EDMD_MPC is
  Port (
    clk          : in  std_logic;
    reset        : in  std_logic;
    data_input   : in  std_logic;
    amp_serial   : out std_logic;
    freq_serial  : out std_logic;
    start        : out std_logic;
    finish       : out std_logic
  );
end Deep_EDMD_MPC;

architecture Behavioral of Deep_EDMD_MPC is

  component clk_wiz is
    Port (
      clk_in1 : in std_logic;
      clk_out1 : out std_logic;
      reset : in std_logic;
      clk_out2 : out std_logic
    );
  end component;

  -- Component declaration for de_serializer
  component de_serializer is
```

```
  Port (clk:       in std_logic;
     reset:     in std_logic;
     data_input: in std_logic;
     clk_data: out std_logic;
     reset_out: out std_logic;
     sample: out fix_point;
     label_SN: out std_logic
      );
end component;

-- Component declaration for Adaptive_controller_top_block
component Adaptive_controller_top_block is
  Port (
     Internal_clk   : in  std_logic;
     clk_input      : in std_logic;
     reset        : in  std_logic;
     input_data   : in  fix_point; -- Vector of input data
     label_SN     : in  std_logic;           -- 32-bit energy output
     control_signal_fs: out fix_point_vector(0 to 1);
     done_out : out std_logic
 );
end component;

-- Component declaration for serializer
component serializer is
  Port (
    clk             : in std_logic;
    control_signal_fs : in fix_point_vector(0 to 1);
    done_in          : in std_logic;
    amp_serial       : out std_logic;
    freq_serial      : out std_logic;
    start           : out std_logic;
    finish          : out std_logic
  );
end component;

-- Signals to connect the blocks
signal reset_out          : std_logic;
signal sample             : fix_point;
signal label_SN           : std_logic;
```

```
  signal control_signal_fs  : fix_point_vector(0 to 1);
  signal done_out           : std_logic;
  signal clk_data_sampling : std_logic;
  signal internal_clk       : std_logic;
  signal clk_input         : std_logic;

begin

  -- Instance of the de_serializer
  U1: de_serializer
    port map (
      clk        => clk_input,
      reset      => reset,
      data_input => data_input,
      clk_data   => clk_data_sampling,
      reset_out  => reset_out,
      sample     => sample,
      label_SN   => label_SN
    );

  -- Instance of the Adaptive_controller_top_block
  U2: Adaptive_controller_top_block
    port map (
      clk_input         => clk_data_sampling,
      Internal_clk      => internal_clk,
      reset             => reset_out,
      input_data        => sample,
      label_SN          => label_SN,
      control_signal_fs => control_signal_fs,
      done_out          => done_out
    );

  -- Instance of the serializer
  U3: serializer
    port map (
      clk              => internal_clk,
      control_signal_fs => control_signal_fs,
      done_in          => done_out,
      amp_serial       => amp_serial,
      freq_serial      => freq_serial,
```

```
      start           => start,
      finish          => finish
    );
   U4: clk_wiz
    port map (
     clk_in1       => clk,
     clk_out1      => internal_clk,
     reset         => reset,
     clk_out2      => clk_input
     );

end Behavioral;
```