Hardware Accelerated Brain-Computer Interfaces for Real-Time Neural Decoding

Rajeev Bhavin Botadra

A thesis
submitted in partial fulfillment of the
requirements for the degree of

MASTERS OF SCIENCE

at the

UNIVERSITY OF WASHINGTON

2025

Committee:

Scott Hauck

Amy Orsborn

Program Authorized to Offer Degree:

Department of Electrical & Computer Engineering

University of Washington

Abstract

Hardware Accelerated Brain-Computer Interfaces for Real-Time Neural Decoding

Acknowledgments

This work is supported by NSF Grant No. 2117997 and conducted in collaboration with the Washington National Primate Research Center.

I owe my sincerest gratitude to Professor Scott Hauck for his guidance and support throughout my time at the University of Washington. Our conversations were instrumental in shaping this research and helping me forge ahead when the path was unclear.

I would like to thank Leo Scholl, Trung Le, and Hao Fang for their guidance in understanding the applications of this work to fundamental neuroscience, as well as their expertise in computational neuroscience for the consideration of the decoding algorithms used in this system.

I would also like to thank ChiRui Chen and YanLun Huang for their help in synthesizing, testing, and optimizing multiple decoding kernels using the HLS4ML, Vitis HLS, and PYNQ pipelines. Thank you to Ryan Canfield for helping set up the Neuropixels hardware and run simulated closed-loop tests. Thank you to Xiaohan Liu for providing useful documentation on configuring the HLS4ML and Vitis development environments.

I would also like to thank Professor Amy Orsborn for sharing her expertise and offering valuable insights into the system's design considerations and broader impact. Special thanks to Professor Shih-Chieh Hsu for introducing me to the A3D3 community, and to Trung Le and Jingyuan Li for their collaboration in overseeing A3D3 initiatives at the University of Washington.

I am very grateful to my family, especially my mother, for their support and understanding through this journey. I also express my heartfelt gratitude to my partner, Christina Liu, for her love, patience, and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

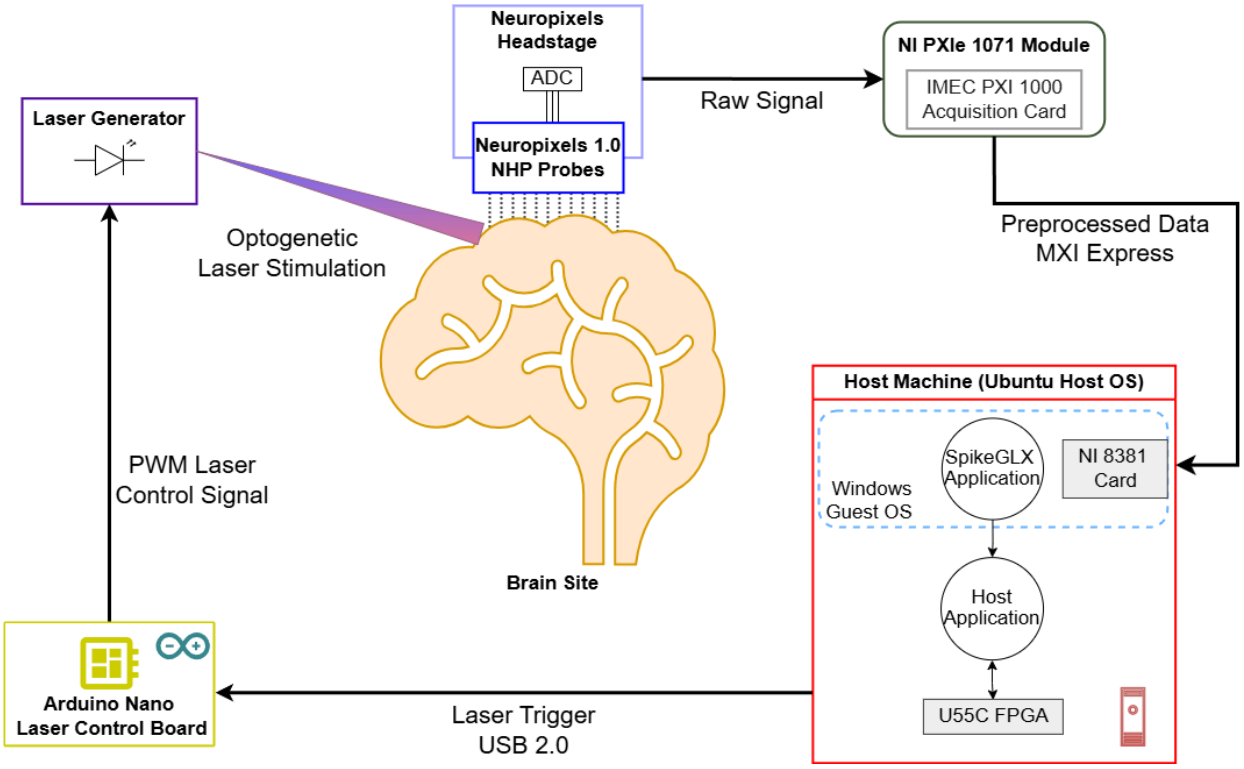

Figure 1.1: Overview of the closed-loop BCI system.

Brain-Computer Interfaces (BCIs) encompass systems that translate neural activity into meaningful outputs, enabling direct communication between the brain and external devices. With advances in high-resolution neural signal recording, modern BCIs have evolved into cornerstone technologies for neuroscience, neuroprosthetics, and neurotherapy. However, the need to record, process, and respond to large volumes of data with low latency and high accuracy imposes challenging constraints on BCI system architecture, including computational

efficiency and data throughput.

This work presents a full-stack co-design of a closed-loop BCI system, integrating a high-density neural recording pipeline with low-latency processing on reconfigurable hardware and efficient inter-process communication between software and hardware subsystems. The system is designed to operate in a closed loop with the brain, where neural activity is recorded, decoded in real time, and used to generate feedback (whether visual, tactile, or electrical) to influence future brain states or behaviors. However, neural signals are non-stationary and often noisy, requiring sophisticated decoding algorithms to capture brain dynamics. Such algorithms are computationally demanding and limit the system's capabilities when employed on traditional hardware architectures, such as CPUs and GPUs. To address decoding model complexity and low-latency execution, the system deploys the Latent Factor Analysis via Dynamical Systems (LFADS) model accelerated by an FPGA (C. et al. 2018). Programmable logic enables extremely efficient and pipelined execution of the decoding model, allowing the system to respond to neural activity in real-time.

The system conducts high-density neural data acquisition using Neuropixels 1.0 probes to capture data from hundreds of channels simultaneously with a spatial resolution on the order of μm. The corresponding SpikeGLX software interfaces with the acquisition system and streams data into buffers on a host machine in real time. The data is then formatted into the expected format for LFADS and transferred to the FPGA over PCIe using Xilinx Runtime (XRT) drivers. Once transferred to the FPGA, the neural data is passed through a quantized LFADS kernel that executes the LFADS decoder in pipelined stages. The decoder produces an estimate of the brain state, which can then be used to drive multiple downstream tasks, which can be varied based on the experiment design. The host machine then evaluates the brain state to signal an Arduino Nano board controlling the optogenetic laser. When a stimulation sequence is triggered by the host, the Arduino board sends a Pulse-Width Modulated (PWM) signal to directly turn the laser on and off. The generated laser pulses are then routed to the optogenetic brain site using a fiber optic cable.

Key criteria for evaluating the system include data transfer latency, decoding throughput, power consumption, and inference accuracy. Our results demonstrate that the deployed decoding algorithm maintains high fidelity in neural decoding even after significant quantization. Additionally, the system achieves sub-millisecond inference latency and a < 10ms end-to-end latency, making it capable of responding to highly active brain regions in real-time while also maintaining modest power consumption compared to CPU and GPU architectures. These properties make the system well-suited for real-time closed-loop operation.

The key contribution of this work is not only the design and deployment of a reliable silicon testbed for real-time neural experiments, but also a broader codesign approach

between the hardware and software systems within the BCI for accelerating neural decoding algorithms. For instance, future iterations of this work can explore the adaptability of the pipeline to alternative neural recording modalities such as EEG, ECoG, and EMG. Another promising direction of research lies in the deployment of other neural decoding algorithms, such as transformer-based models. Transformers, known for their scalability and capacity to model long-range dependencies, have shown recent promise in modeling neural population activity. However, their deployment on FPGAs remains an ongoing challenge, especially under constraints of available memory and logic resources. Finally, though the system has been validated in both simulated and emulated environments, it does not completely address real-world challenges such as signal drift. Accommodating brain plasticity and experiment-to-experiment variability is an important requirement in deploying the silicon testbed across multiple experiments, and while some proposals to address the data shift problem have been introduced, it remains future work to test and deploy those ideas.

In summary, this work presents a comprehensive integration of software and hardware to form a practical, low-latency BCI system capable of real-time neural decoding using high-density recordings. Through a co-design of both the hardware and software subsystems, this work demonstrates the feasibility of deploying advanced decoding models under tight timing and accuracy constraints. The resulting framework not only advances the boundaries of existing integrated BCI platforms but also lays the groundwork for future developments in closed-loop neuroscience and neuroengineering.

Chapter 2

Neuroscience Background

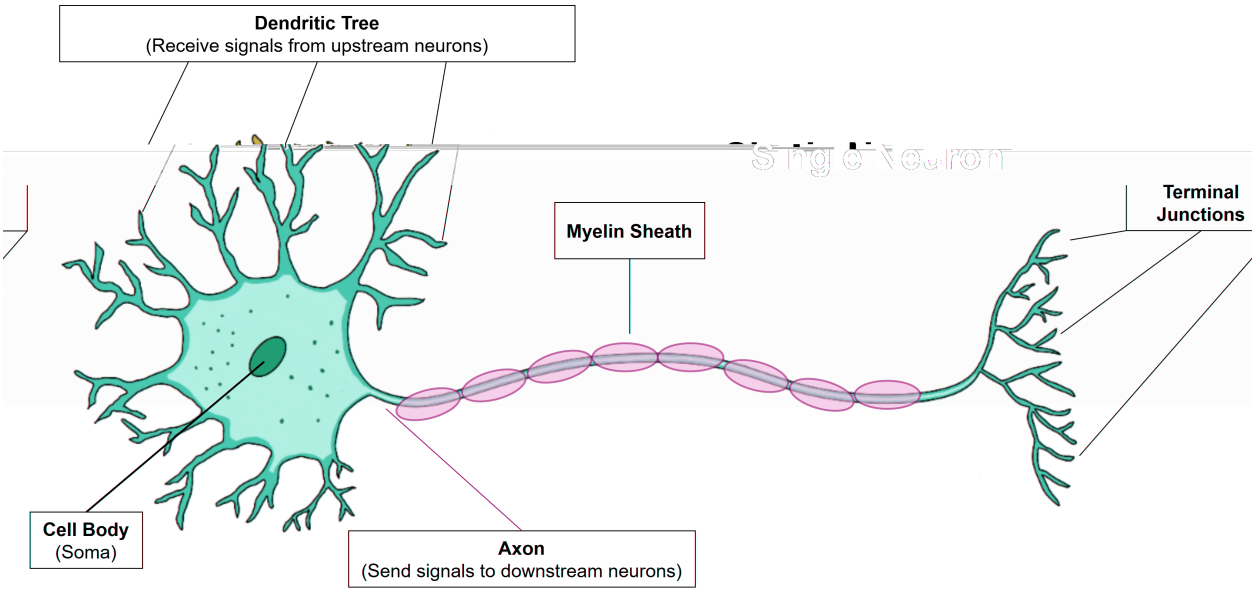

Figure 2.1: The physiology of a neuron, including the cell body, axon, dendritic tree, and myelin sheath.

2.1 Physiology of a Neuron

Neurons are the fundamental unit of the nervous system, responsible for communicating electrochemical signals between the body and the brain. These signals, known as action potentials, propagate through a network of connections between neurons. The transmission and reception of this signal is mediated by moving ions such as Sodium (Na^+), Potassium (K^+), and Calcium (Ca^{2+}) across the cell wall of the neuron, thereby altering intracellular

and extracellular electric fields[1]. The structure of a typical neuron can be represented by three key components:

1. **Soma**: The cell body housing the cellular organelles of the neuron.
2. **Dendritic Tree**: Connections to presynaptic ("upstream") cells to receive action potentials.
3. **Axon**: The primary connection to postsynaptic ("downstream") cells to transmit action potentials.

The axon of one cell and the dendrite of the next are connected via terminal junctions. At equilibrium, the postsynaptic cell maintains a potential difference across its cell wall with a higher concentration of negative charges inside and positive charges (ions) outside the cell. When an action potential travels down the presynaptic axon, it triggers the release of neurotransmitters at the junction. The postsynaptic dendrite receives the neurotransmitters and opens specific ion channels (such as Na+), allowing ions to flow across the cell membrane and create a local depolarization. If the local depolarization voltage exceeds a cell-specific threshold (commonly around -50 mV), the depolarization propagates down the postsynaptic dendrite to the soma and causes the cell to depolarize, or *fire*, producing a consequent action potential known as a *spike*[2]. During the spike, the cell experiences rapid depolarization (on the order of 2ms), resulting in a strong positive membrane potential. This spike signal travels down the axon of the postsynaptic cell to downstream neurons, once again triggering downstream spikes depending on the signal strength and the cell-specific sensitivities. Following the rapid depolarization, the cell enters a repolarization phase, and a refractory period ensues, during which the cell's membrane potential returns to its resting potential.

The changes in intracellular action potentials have a measurable effect on the extracellular electric field as the ions flow across the cell membrane through dedicated ion channels. These signals—or more often, their superposition— are detected by probes placed outside the cells and allow for the study of local brain activity.

2.2 The Hodgkin-Huxley Model

The Hodgkin-Huxley (HH) model is a mathematical representation of a neuron that captures neuronal action potentials by modeling the ion channels of the neurons in a circuit. Originally

[1]This is a general description of neuronal physiology that does not capture the vast variation in behavior, characteristics, and roles that neurons throughout the brain and body play. For a more nuanced dive into the diversity of neurons and their functional specializations, refer to (Lodato et al. 2015).

[2]The terms "Action Potential" and "Spike" will be used interchangeably hereon.

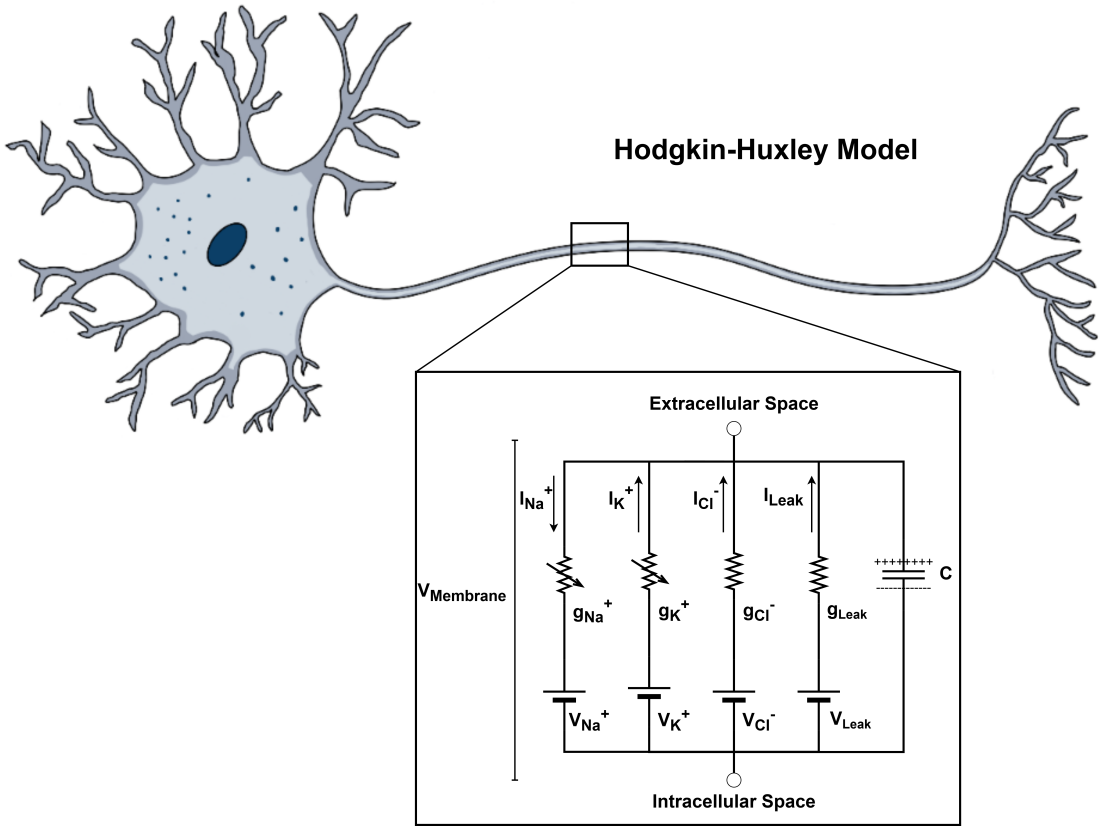

Figure 2.2: The Hodgkin-Huxley model of action potential generation in the axon. The model approximates the ion channels in the membrane with individual circuit paths.

developed from voltage-clamp experiments on the squid giant axon (Hodgkin et al. 1952), the HH model characterizes the dynamics of the neuronal membrane potential as governed by voltage-dependent ion channels by modeling them as nonlinear conductances, primarily Sodium (Na^+), Potassium (K^+), and a leak current.

The core assumption of the HH model is that the neuronal membrane behaves as an electrical circuit composed of capacitive and conductive elements. The membrane potential $V(t)$ evolves according to the following differential equation:

$$C_m \frac{dV}{dt} = -(I_{Na} + I_K + I_{Leak}) + I_{External} \tag{2.1}$$

Where:

- C_m is the membrane capacitance.
- $I_{Na}, I_K, I_{Leak}, I_{External}$ are the currents for the Sodium and Potassium ion channels, leakage, and external sources, respectively.

Each ionic current is modeled by Ohm's law:

$$I_{Na} = m^3 h G_{Na}(V - E_{Na}) \tag{2.2}$$

$$I_K = n^4 h G_K(V - E_K) \tag{2.3}$$

$$I_{Leak} = G_{Leak}(V - E_{Leak}) \tag{2.4}$$

Where:

- G_{Na}, G_K, G_{Leak} are the conductances.
- E_{Na}, E_K, E_{Leak} are the reversal potentials (thresholds).
- m, n, h are gating variables representing ion channel states.

The gating variables evolve according to voltage-dependent first-order ordinary differential equations:

$$\frac{dm}{dt} = \alpha_m(1 - m) - \beta_m m \tag{2.5}$$

$$\frac{dh}{dt} = \alpha_h(1 - h) - \beta_h h \tag{2.6}$$

$$\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n \tag{2.7}$$

Where α and β are empirically derived rate constants that govern the opening and closing of channels. The HH model reproduces the shape of action potentials with high accuracy and can be used to predict neural dynamics when combined with a state estimation model such as the Unscented Kalman Filter (see chapter 3, Neural Signal Decoding).

2.3 Brain Signals

An action potential is a binary signal: it either occurs and propagates the signal downstream or it does not occur. Furthermore, the amplitude or duration of the spike does not encode any information – that is, a postsynaptic neuron is invariant to the signal magnitude and will fire as long as the amplitude of the spike is greater than its threshold voltage. Instead, neurons encode information by modulating the frequency of their action potentials.

The most fundamental representation of neural activity over time is the spike train, a sequence of action potentials recorded from a single neuron over time. Spike trains are typically derived by binarizing the raw voltage trace, with each time bin designated as "1" if a spike occurs and "0" otherwise. This binary signal representation abstracts away the

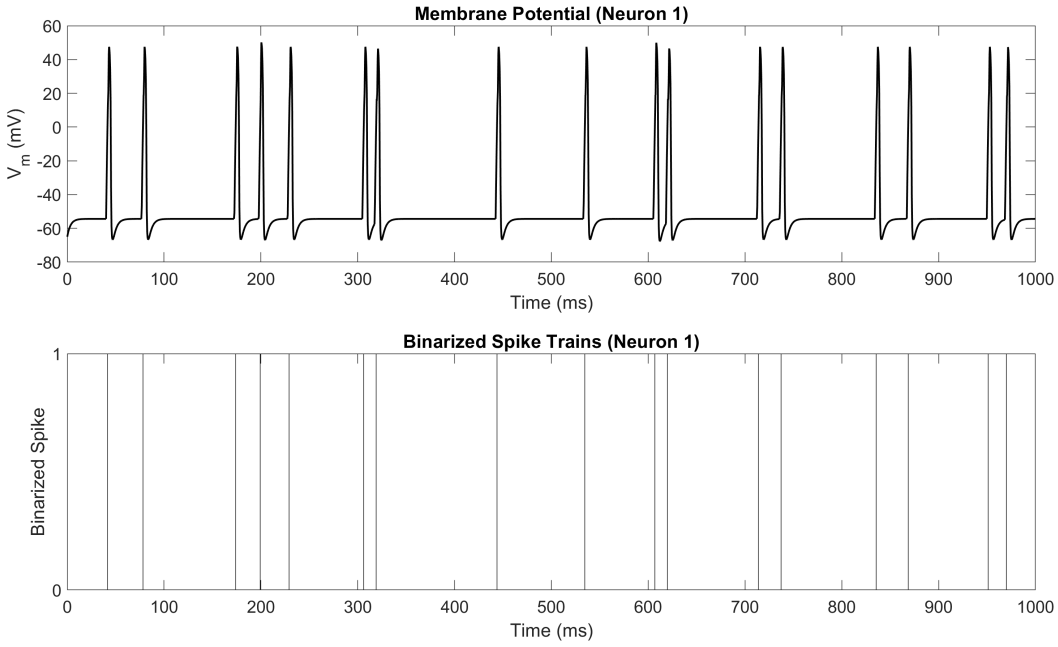

Figure 2.3: Raw membrane potential and binarized spikes for a single neuron simulated with the Hodgkin-Huxley model. Spike threshold set to 20mV.

shape and amplitude of spikes, focusing solely on their timing or firing rate (average number of times a neuron has fired over a certain time window).

Extracellular recordings often contain a mixture of signals originating from multiple neurons, along with noise from instrumentation and the environment. These recordings form a *Local Field Potential* (LFP), a low-frequency signal reflecting the summed synaptic activity in a local region. To isolate spiking activity, signal processing techniques such as band-pass filtering (e.g., 300–3000 Hz) are applied to enhance the Signal-to-Noise Ratio (SNR) by rejecting low-frequency background signals. Spike detection is then performed by applying a threshold to the filtered signal. There are multiple approaches to setting the threshold voltage for a signal. One could set the threshold manually, given prior knowledge about the expected signal amplitude for the spikes, or by using statistical properties of the signal, for example, using a multiple of the estimated standard deviation of the noise. The choice of threshold directly influences spike detection fidelity and firing rate estimates and can significantly affect downstream decoding performance.

To visualize spike trains from multiple trials in an experiment, *raster plots* are commonly used. Each row in a raster plot represents a single trial or recording site, with tick marks denoting spike times. Raster plots are particularly useful for visualizing neural response when a stimulus is presented, as one can align the data to the time of stimulus presentation and

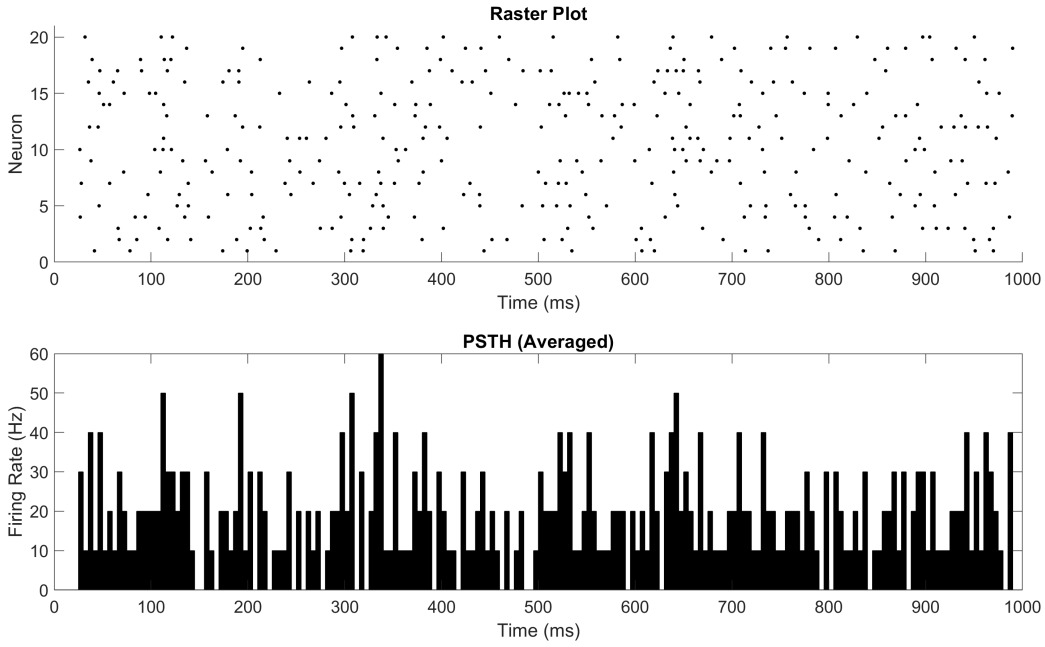

Figure 2.4: Raster plot of activity of 20 simulated Hodgkin-Huxley neurons and Peri-Stimulus Time Histogram averaged across all cells.

visualize firing rate activity before and following the stimulation. Furthermore, a *Peri-Stimulus Time Histogram* (PSTH) can be used to display the firing rate versus time numerically rather than visually by averaging the firing counts of a neuron over time bins, providing a useful visualization of a neuron's firing frequency over time following an external reference event (e.g. stimulus onset). Generally, neural decoding models act on binarized spike trains or LFPs directly, while raster plots and PSTHs are useful for visualizing the neural activity on a channel for experiments with external stimulation.

2.4 Recording Modalities

Different modalities used to measure electrical activity generated by neurons and muscle fibers offer tradeoffs among spatial resolution, temporal resolution, invasiveness, and cost. Choosing a recording technique is dependent on the experimental or clinical context, scale (small study or commercialization), and constraints on patient risk.

1. **Patch Clamp Recordings**: An intracellular recording technique that forms a seal between the neuronal membrane and a laboratory micropipette to directly observe the movement of ions across the membrane. Provides the best possible spatial and temporal resolution but is highly invasive and impractical for in vivo applications.

2. **Intracortical Recordings**: Electrodes penetrate the cortical surface to record action potentials from individual or small clusters of neurons. Offer great spatial resolution (on the scale of tens of microns) and high temporal resolution (on the order of milliseconds), enabling real-time decoding of neural activity. Still, an invasive and expensive surgery is required to install the electrode implants.

3. **Electroencephalography** (EEG): A noninvasive method that places electrodes on the scalp and measures aggregate neural activity. Great temporal resolution (on the order of milliseconds) but poor spatial resolution due to signal attenuation through the skull. However, it is low-cost, portable, and viable for a large variety of experimental and clinical applications.

4. **Electrocorticography** (ECoG): Electrodes are placed directly on the cortical surface, have higher spatial resolution than EEG, but are also more invasive. Still, less invasive and damaging to brain tissue than intracortical recordings.

5. **Electromyography** (EMG): Records electrical activity from muscles rather than neurons.

6. **Surface Electromyography** (sEMG): Electrodes on the skin above muscles, providing a noninvasive and low-cost method for observing muscle activity. It has good temporal resolution but suffers from poor spatial resolution due to crosstalk from adjacent muscles. Widely used in prosthetics.

Table 2.1: Comparison of Electrophysiological Recording Modalities

Modality	**Spatial Resolution**	**Temporal Resolution**	**Invasiveness**	**Cost**
Patch Clamp	Very High	Very High	Very High	High
Intracortical	High	High	High	High
EEG	Low	High	Very Low	Low
ECoG	Medium	High	Medium	Medium
sEMG	Low	High	Very Low	Low

2.5 Brain-Computer Interfaces

Brain-computer interfaces (BCIs) are systems that establish a mode of communication between the brain and an external device, bypassing the natural pathways of the nervous system. By decoding neural activity and translating it into actionable commands, BCIs

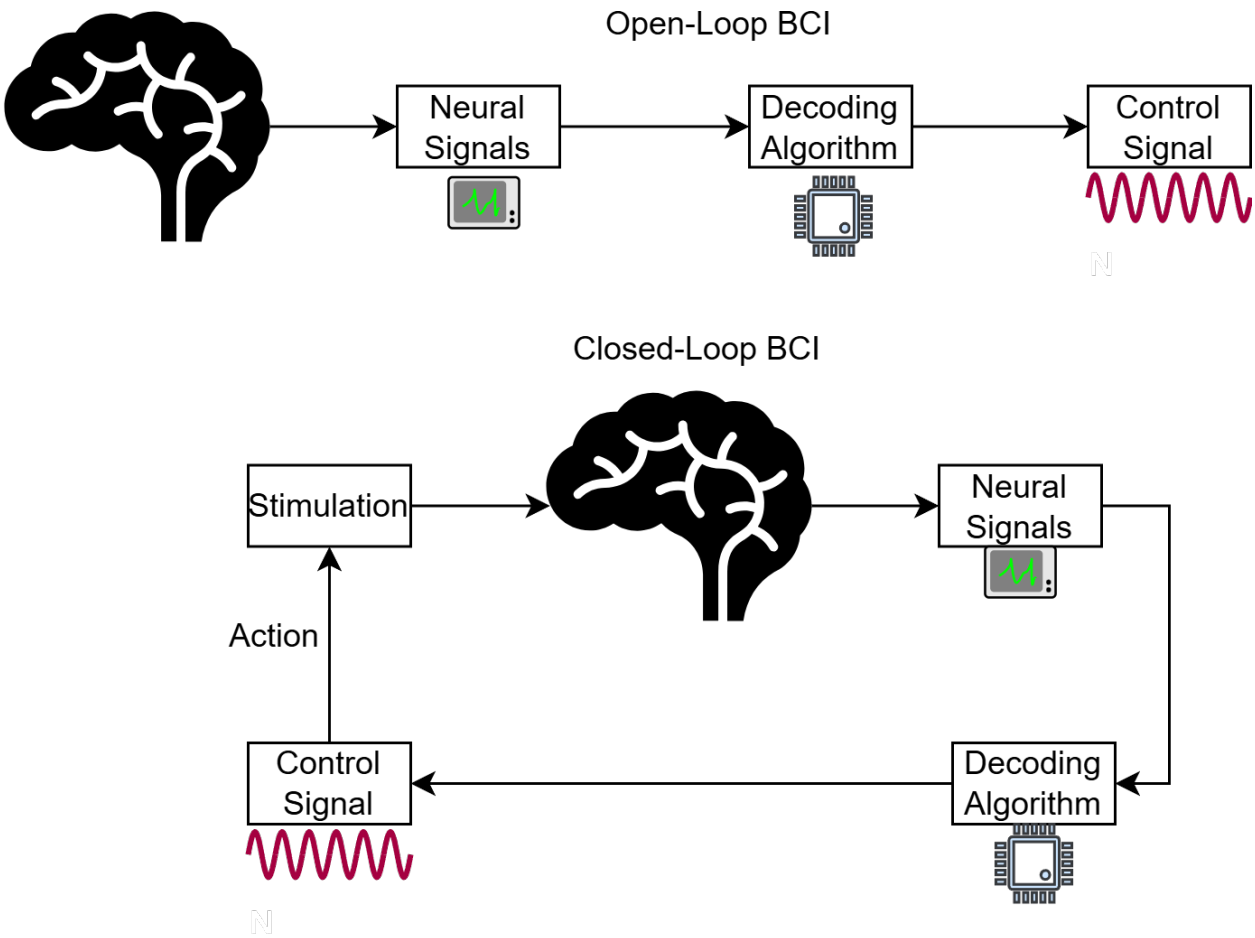

Figure 2.5: Abstract overview of open-loop and closed-loop BCIs. Note that the stimulation can be direct (electrical) or indirect (sensory).

enable many critical applications ranging from motor prostheses to the study of fundamental neuroscience (Vilela et al. 2020). The brain can be abstracted as a dynamically evolving system interacting with its environment, with inputs (in the form of sensory stimulation, such as sight) and outputs (actions to control the physiological processes). A BCI extends this model, interacting with the brain to create an artificial pathway between the brain and the world.

BCIs generally fall into two categories, Open-Loop BCIs and Closed-Loop BCIs, referring to the flow of information between the brain and the world. Open-loop BCIs have a unidirectional information pathway with the brain, either decoding neural activity without any corresponding stimulation or stimulation without any corresponding decoding. In some instances, encoder/decoder models are trained in a closed loop to better learn local neural dynamics and then implemented in an open loop. Meanwhile, closed-loop BCIs have a bidirectional flow of information with the brain, both decoding neural activity and

stimulating the brain, creating a feedback mechanism. Feedback can take multiple shapes, including visual, auditory, haptic, or electrical. For instance, a BCI to control a cursor using muscle activity (sEMG) provides visual feedback to the subject (the cursor moving on a screen). Meanwhile, a BCI designed to perform Deep Brain Stimulation (DBS) to counteract Alzheimer's symptoms provides electrical impulses directly to the brain. The feedback mechanism can substantially improve the subject's ability to control the prosthetic due to the plastic and adaptive nature of the brain.

While there is no universal metric for evaluating all BCI systems due to their varied applications and goals, key areas that impact performance include:

1. Decoding Accuracy: Decoder's ability to model brain intent.
2. System Latency: Time taken to process the input signal from the brain and generate an output.
3. Power Consumption: Total power consumed by all system components, a critical consideration for embedded BCIs.

For systems with direct user control (e.g. using sEMG) for cursor control, system latency plays a critical role in defining the user experience with the BCI. Latencies close to or exceeding the human reaction time (typically greater than 100ms) significantly degrade the user's ability to control and adapt to the BCI. Therefore, minimizing computational and transmission delays is essential to ensure system responsiveness and usability. Additional important metrics for real-world applications include stability across multiple sessions of use and robustness to neural variability and drift. The important considerations in designing the system of this work are detailed below (see chapter 4, System Implementation).

2.6 Optogenetics

Optogenetics is a neurostimulation technique that enables precise control of neuronal activity using genetically encoded, light-sensitive ion channels. By doping populations of neurons with specific viral agents experimenters can excite or inhibit neural activity by introducing specific wavelengths of light. The Channelrhodopsin-2 (ChR2) channel is more sensitive to lower-wavelength light (e.g., blue light) and allows Sodium ions into the intracellular space, depolarizing the cell and triggering a spike. The Halorhodopsin (NpHR) channel is sensitive to higher-wavelength light (e.g., red/yellow light) and allows Chlorine ions into the cell, hyperpolarizing the cell and inhibiting spiking activity (Boyden et al. 2005).

The laser is directed to a fixed brain region through a fiber-optic cable and is activated and deactivated with millisecond-scale precision to finely control spiking activity in the region.

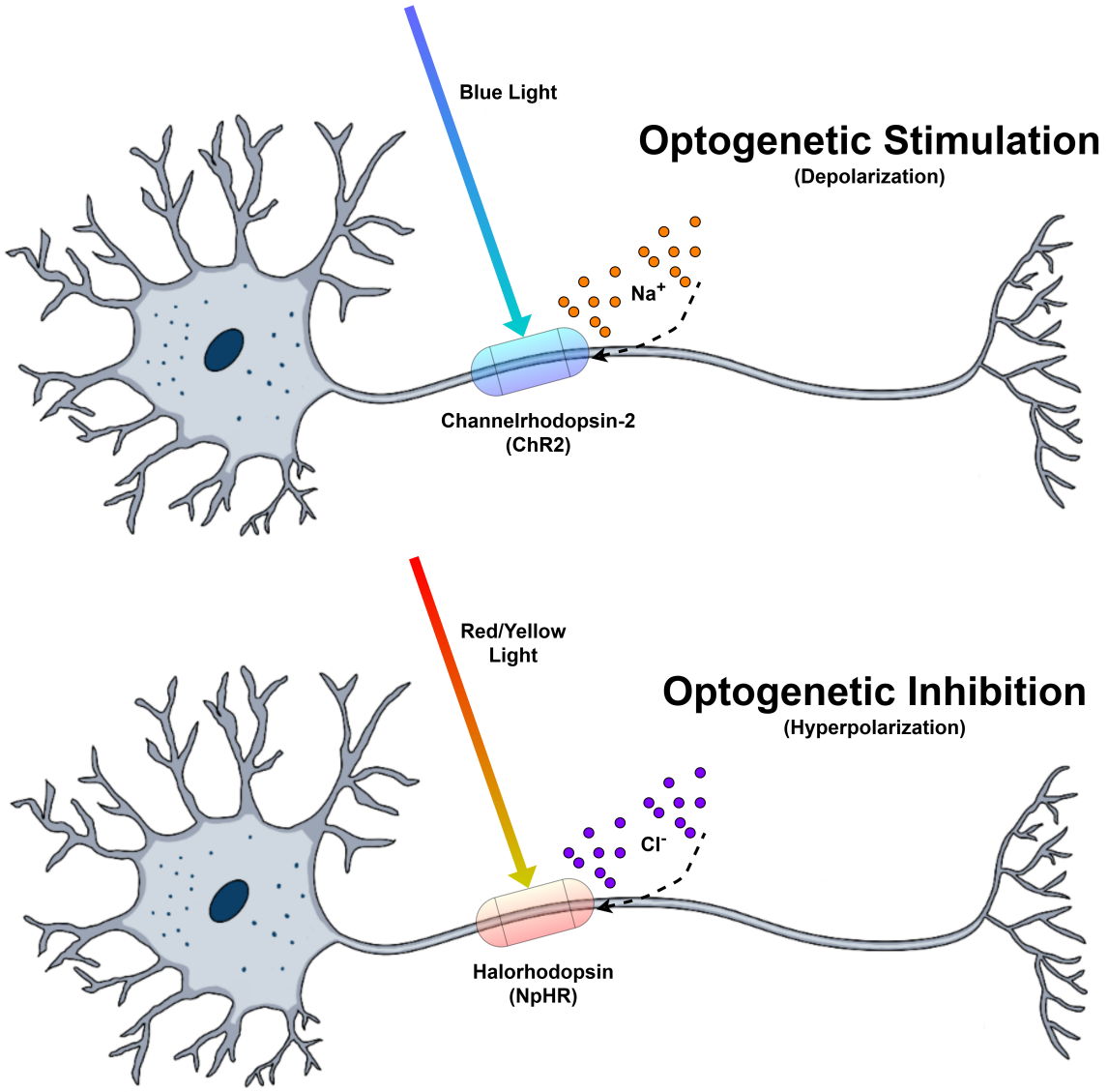

Figure 2.6: Optogenetic stimulation and inhibition of a neuron.

This high degree of temporal precision makes optogenetic stimulation a valuable technique for stimulating brain regions and probing causal relationships between neural circuits and behavior in this study. Optogenetics has been used widely in academia for the study of fundamental neuroscience, such as the mechanisms behind memory retention. However, some limitations of this technique include its invasiveness, need for genetic modification, and the limited cortical depth that can be reached by the stimulant. For this work, optogenetics was a readily available option for stimulation due to its use in the existing experimental apparatus at the aoLab of the University of Washington.

Chapter 3

Neural Signal Decoding

3.1 Type of Decoding Models

Neural signal decoding is the task of reconstructing behavioral, cognitive, or physiological variables from observed neural activity. This process is crucial to both the study of fundamental neuroscience, where decoding provides insight into the functional representations of neural populations, and applied domains such as BCIs, where decoding enables the direct translation of brain activity into external control signals. Based on observed spikes, local field potentials, or other electrophysiological signals, the decoding model aims to extract latent information such as intended movement, sensory stimuli (predicting the stimulus that triggered the signal), or internal cognitive states.

To achieve this, neural decoders create state estimation frameworks that assume some latent process generates the observed neural dynamics. This latent process is unobserved and unknown, and the decoder forms some hypotheses (either given a prior model of the neuron circuit or by learning one using machine learning) of the latent dynamics to map the observed activity to estimates of the underlying state.

3.1.1 Linear Models

Linear state estimation models, such as the Kalman filter and Wiener filter, assume that both the latent state dynamics and the neural observations are governed by linear relationships corrupted by Gaussian noise. These models are widely used due to their simplicity to implement in real-time pipelines (Malik 2010). However, linear models struggle to capture the nonlinearities and high-dimensional interactions inherent in real neural data. Even the simplest case of modeling the behavior of a single neuron contains nonlinear dynamics between the ion channel currents and the membrane potential, as seen in the Hodgkin-Huxley model.

Their limited expressivity makes them nonviable solutions to deploy in real-time BCIs and motivates the study of more complex, nonlinear models.

3.1.2 Nonlinear Models

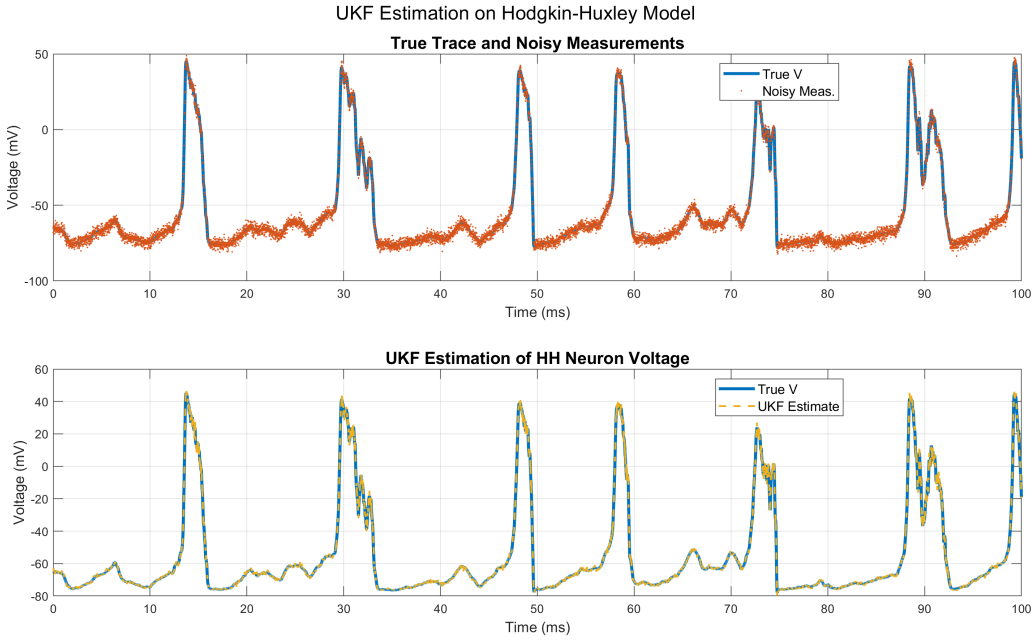

Figure 3.1: Plot of the original and predicted activity of a neuron using an Unscented Kalman Filter using the Hodgkin-Huxley model to evolve state dynamics.

Nonlinear decoding models, such as the Unscented Kalman Filter (UKF), are far more expressive in their ability to capture complex latent dynamics than their linear counterparts. Advances in recording technology, including higher spatial resolutions and sampling frequencies, have also created large volumes of high-dimensional data to be leveraged by ML methods. Recurrent Neural Networks (RNNs), for instance, are capable of modeling temporally structured signals and capturing nonlinear dependencies across time. RNN variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks have been successfully applied to neural decoding tasks, and these architectures form the basis of many modern neural decoding systems. The following sections describe specific deep learning architectures used in neural decoding, beginning with RNNs, then expanding into self-supervised models such as autoencoders and variational autoencoders, and culminating in the decoder used in this work: the Latent Factor Analysis via Dynamical Systems (LFADS) model.

3.1.3 Offline and Online Decoding

Decoding approaches can broadly be divided into offline and online (or real-time) paradigms. Offline decoding refers to the retrospective analysis of neural data after acquisition. While offline methods are widely used in experimental neuroscience for exploring the representational content of recorded signals, they cannot interact with the brain and influence its dynamics in real-time. In contrast, online decoding involves estimating behavioral or cognitive states in real-time, concurrently with neural data acquisition. This paradigm is essential for a closed-loop system, which adaptively interacts with the brain through feedback. Online decoding allows experiments to probe and test the functions of neural circuits and their role in physiological processes such as learning or memory retention. For example, when studying the role of a sleep spindle neuron, experimenters can inhibit activity when the decoder predicts the neuron will become active and study the consequent cognitive impacts. Real-time decoding models are particularly critical in BCIs, where the real-time translation of neural intent into action is required for effective performance.

3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for modeling sequential and time-dependent data. Unlike feedforward networks, which assume independent inputs, RNNs incorporate feedback connections that allow them to maintain and update a hidden internal state over time. Through this hidden state, the outputs of the RNN at timestep T are influenced by all previous inputs $[x_0, \cdots, x_{T-1}]$. This makes them particularly well-suited for decoding tasks in neuroscience, where the underlying neural signals are generated by dynamic processes that evolve over time and exhibit strong temporal correlations.

Mathematically, a standard RNN is defined by the recurrence:

$$h_t = \sigma(W_h h_{t-1} + W_{xh} x_t + b_h) \tag{3.1}$$

$$y_t = W_{hy} h_t + b_y \tag{3.2}$$

where x_t is the input at time t, h_t is the hidden state, y_t is the output, and $\sigma(\cdot)$ is a nonlinear activation function. The weight matrices W_{xh}, W_{hh}, W_{hy} and biases b_h, b_y are learned through gradient-based optimization. The hidden state h_t serves as a dynamic memory that aggregates past information, allowing the network to learn mappings from input histories to output sequences.

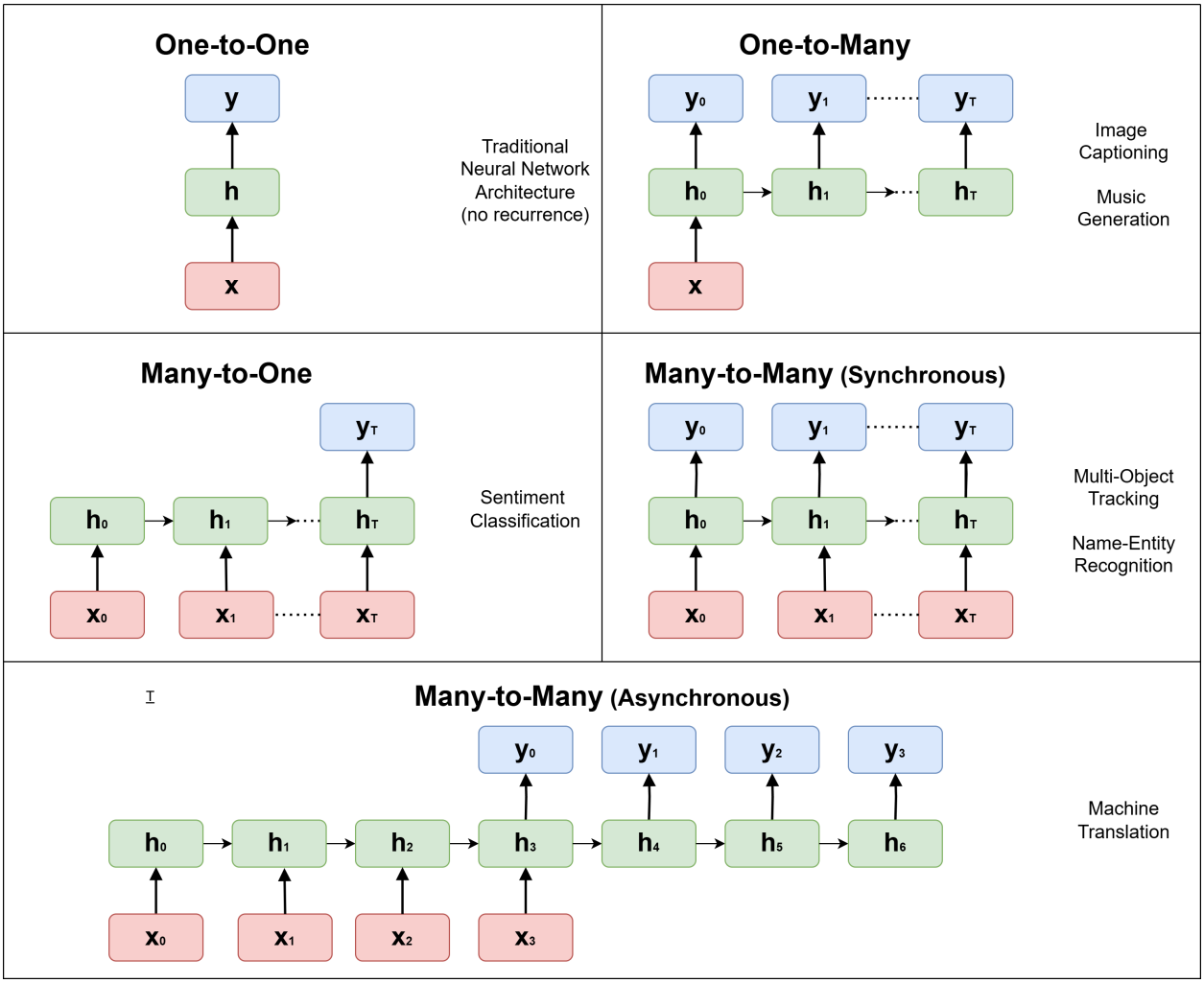

Figure 3.2: Diagram of the various RNN architectures and suitable tasks for each case.

Depending on the relationship between input and output sequences, RNN architectures can be categorized into several distinct types, each suited for different applications.

1. **One-to-One**: Maps a single input to a single output, functioning similarly to a traditional feedforward neural network. While not inherently sequential, it serves as a foundational structure for understanding more complex RNN variants.

2. **One-to-Many**: A single input generates a sequence of outputs. This architecture is commonly employed in tasks such as image captioning, where a fixed-size image input is transformed into a variable-length textual description.

3. **Many-to-One**: A sequence of inputs is processed to produce a single output. Applications include sentiment analysis, where a series of words (input sequence) is classified into a sentiment category (single output).

4. **Many-to-Many** (Synchronous): Both input and output are sequences of equal length, enabling applications like video frame labeling or named entity recognition, where each input element corresponds directly to an output element.

5. **Many-to-Many** (Asynchronous): Processes variable-length input sequences to produce variable-length outputs, making it essential for machine translation and speech recognition, where input and output sequences differ in structure and length (e.g., translating spoken phrases into written sentences).

However, despite their theoretical expressiveness, basic RNNs are limited in their ability to model long-range dependencies in sequences due to issues in vanishing and exploding gradients during training (Pascanu 2013). Gated RNN variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) have been developed to address this issue. These architectures introduce gating mechanisms that regulate the flow of information, preventing gradients from experiencing exponential decay or growth during backpropagation and enabling the network to preserve relevant context over extended durations.

As neural signal decoders, RNNs have been demonstrated to be capable of capturing the complex, nonlinear relationships between sequences of neural activity (e.g., spike trains, local field potentials) and corresponding time-varying behavioral or cognitive states (Tampuu 2019). Because RNNs can perform real-time inference, they are also compatible with closed-loop BCI systems that require online decoding. For instance, in motor decoding, RNNs can be trained to predict limb trajectories from multi-electrode recordings in the motor cortex. Furthermore, RNNs can naturally replicate the state estimation framework of traditional decoding models through their hidden state, which serves as a compressed lower-dimensional representation of neural dynamics. These encoded latent states are more explicitly studied in the generative models discussed next.

3.3 Autoencoders, Variational Autoencoders, and Self-Supervised Learning

The latent states implicitly modeled by RNNs can be more finely studied using generative models such as Autoencoders (AEs) and Variational Autoencoders (VAEs). These architectures use an encoder-decoder framework to produce low-dimensional representations of neural population activity that approximate the latent dynamics within the brain. This representation can be decoded to reconstruct neural activity or perform downstream tasks such as predicting hand movement.

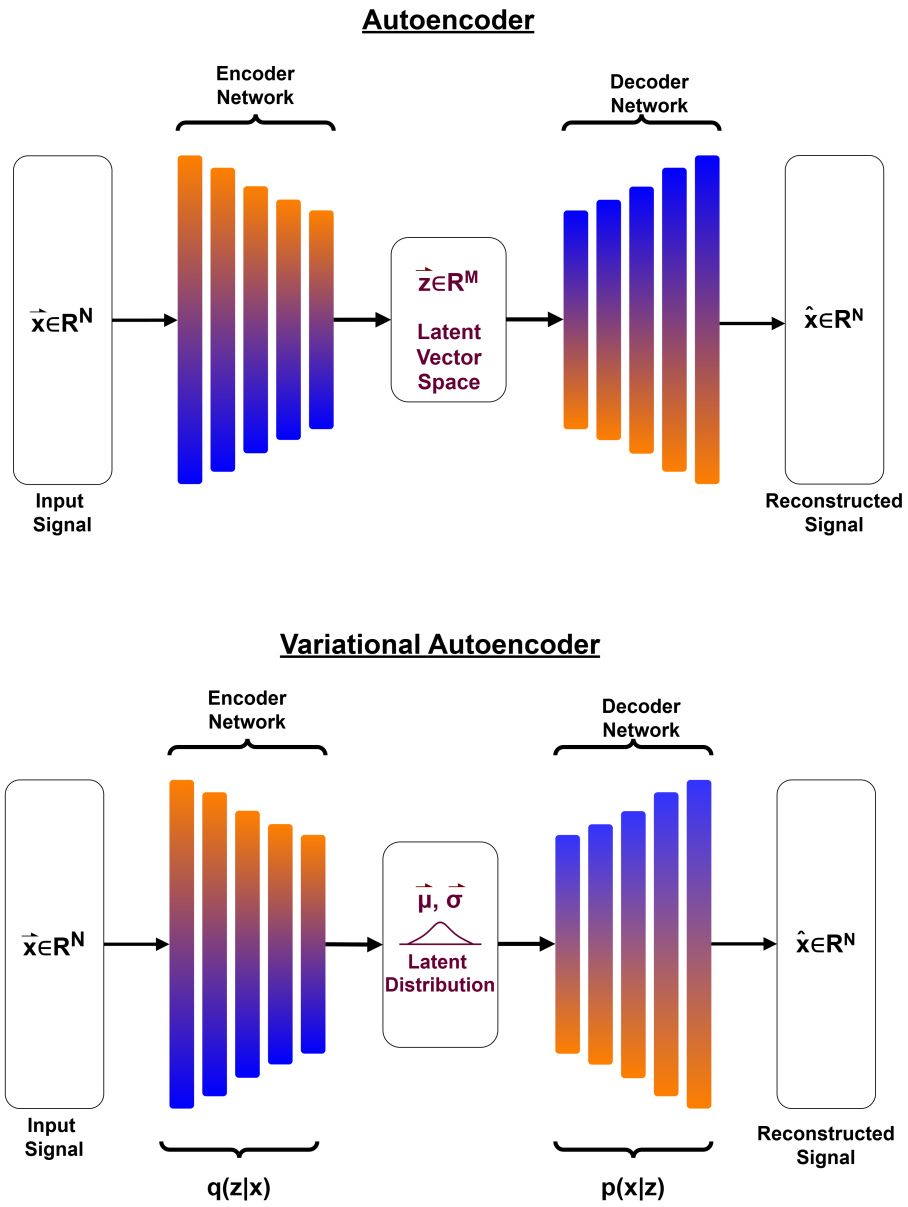

Figure 3.3: The architecture of Autoencoder and Variational Autoencoder networks.

The ability to reconstruct input data gives generative models the advantage of not requiring a dataset of input and output pairs ($\mathcal{D} : \{(X_i, y_i)\}_{i=[1,\cdots,n]}$). Instead, generative models can be trained on an unlabeled dataset ($\mathcal{D} : \{X_i\}_{i=[1,\cdots,n]}$) through Self-Supervised Learning (SSL).

3.3.1 Self-Supervised Learning

SSL encompasses a set of training techniques where supervision is provided by the data itself, rather than externally defined labels. This is particularly valuable in neuroscience, where labeled data is often limited or unavailable and recordings of raw neural signals are far more abundant. The objectives in SSL direct the model to reconstruct the input data with high accuracy and, for time-series data, with temporal continuity. Another key advantage of SSL training is that it allows decoders to generalize across experimental sessions or even different subjects by learning invariant structure from the raw signal itself. This improves the scalability and reusability of neural decoding pipelines and helps to address data shift, a key consideration of BCI design.

3.3.2 Autoencoders

An autoencoder is a type of neural network architecture designed to learn efficient encodings of input data. It consists of two components: an encoder $f_\theta(\cdot)$, which maps high-dimensional input $\mathbf{x} \in \mathbb{R}^D$ to a lower-dimensional latent vector space $\mathbf{z} \in \mathbb{R}^d$ (where $d \ll D$), and a decoder $g_\phi(\cdot)$, which reconstructs the original input from the latent representation. The model is trained by minimizing a reconstruction loss, typically mean squared error or cross-entropy, depending on the input data:

$$\mathcal{L}_{AE}(\mathbf{x}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x} - g_\phi(f_\theta(\mathbf{x}))\|^2 \tag{3.3}$$

Autoencoders provide a means to uncover latent structure in high-dimensional recordings, such as multi-electrode spike trains or local field potentials. However, standard AEs model the latent space through vectors and do not support a probabilistic interpretation of the latent space. This is a concern because deterministic reconstruction of the input reduces the expressivity of the AE and can lead to instances where the model simply recreates signals it has seen during training.

3.3.3 Variational Autoencoders

Variational autoencoders extend the basic AE framework by introducing a probabilistic model of the data. Rather than encoding each input into a single point in the latent vector space, a VAE encodes it into a distribution over latent variables, typically modeled as a multivariate Gaussian. The encoder maps an input $\mathbf{x}$ to the parameters (μ, σ) of a posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$, from which a latent variable $\mathbf{z}$ is sampled. The decoder then reconstructs $\mathbf{x}$ from

this sample. The objective function is defined by the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{VAE}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) \tag{3.4}$$

where D_{KL} denotes the Kullback-Leibler divergence between the learned posterior and a prior (usually standard normal). The first term encourages accurate reconstruction of the input, while the second term regularizes the latent space by penalizing divergence from a prior distribution. This formulation encourages the model to learn a smooth, continuous, and interpretable latent space from which data can be sampled.

In neural decoding, VAEs are particularly advantageous in modeling neural variability and uncertainty. Neural responses are inherently stochastic due to noise, intrinsic biological variability, and activity from distant parts of the brain. VAEs can intrinsically model this stochasticity by adjusting the parameters of the latent variable distributions. The latent spaces learned by VAEs have been shown to align well with low-dimensional neural manifolds observed empirically in population coding studies. For instance, VAEs have been applied to hippocampal recordings to recover latent spatial maps, or to motor cortex data to infer underlying movement plans (C. et al. 2018).

3.4 The LFADS Architecture

The Latent Factors Analysis via Dynamical Systems (LFADS) model is a Variational Autoencoder Recurrent Neural Net (VAE-RNN) architecture that constructs latent representations of causal signals in neural data, and is the decoder model tested and deployed in this work (Sussillo et al. 2016). It uses a VAE to produce latent representations (denoted as latent factors) of the neural data and an RNN to encode and decode signals across a time series. The model assumes that there exists some dynamical system which, along with external perturbations, can describe the behavior of a local region of neurons. It further constrains this dynamical system to express the firing activity of each neuron in the system to be generated by a Poisson random process with a true firing rate λ. These external perturbations represent the influence of activity from other brain regions not directly monitored by the experiment. The brain signals are recorded from extracellular electrodes at a configurable sampling frequency and batched into a $N_C \times T$ matrix corresponding to the number of electrode channels and the number of time bins, respectively. This batched data $X_{1:T}$ is input into the LFADS model, which first produces an encoded latent representation of the input using a bi-directional RNN. The encoder first produces a forward representation of the raw signals X_t from $t = 1$ to $t = T$. Then the backward representation is generated starting from

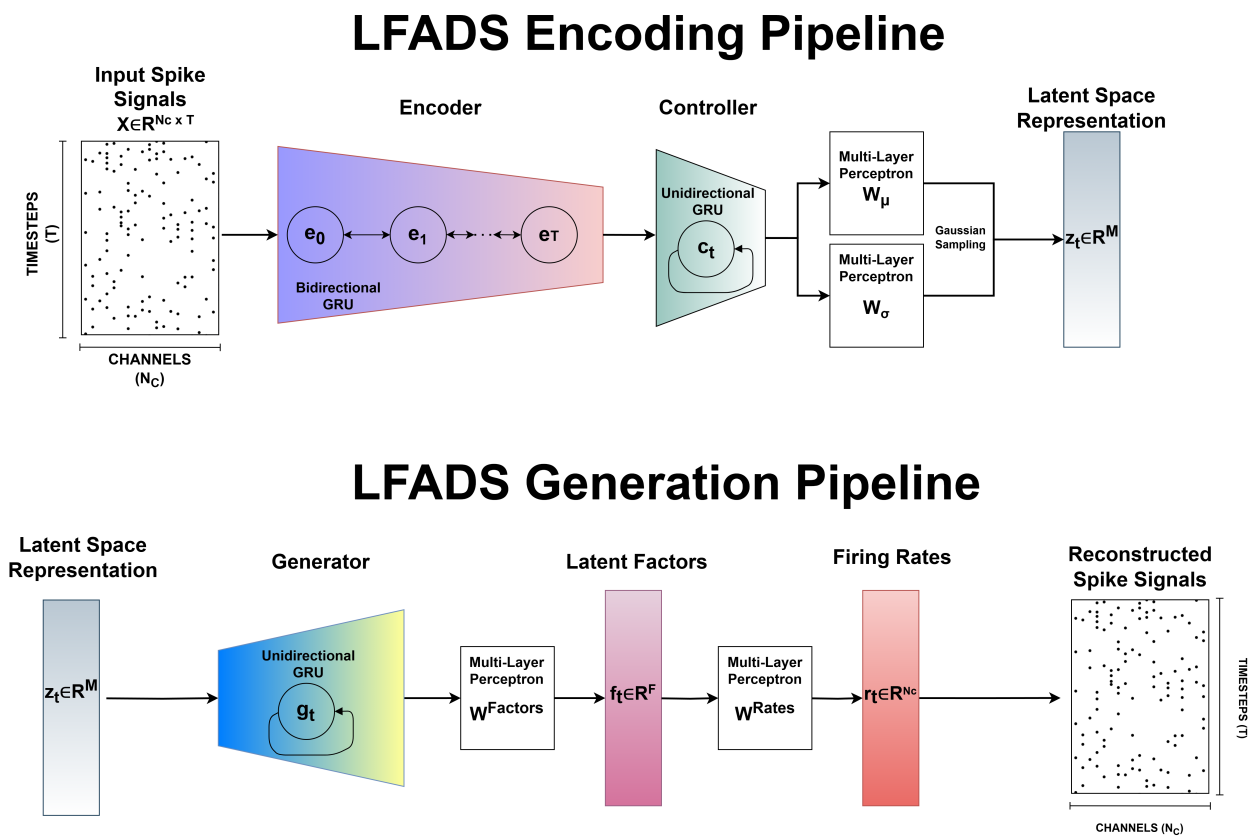

Figure 3.4: The LFADS architecture, a VAE-RNN based state estimation framework.

$t = T$ to $t = 1$. The final encodings (at the last timesteps of $t = T, t = 1$ for the forward and backward encodings, respectively) are concatenated and fed to the controller layer of the network.

$$e_t^b = \text{RNN}^{\text{ENC}}(e_{t+1}^b, x_t) \tag{3.5}$$

$$e_t^f = \text{RNN}^{\text{ENC}}(e_{t-1}^f, x_t) \tag{3.6}$$

$$h = [e_1^b, e_T^f] \tag{3.7}$$

Where the input data is a matrix with N_c columns and T rows, $X \in \mathbb{R}^{N_C \times T}$, and x_t represents the signal at time t across all channels, $x_t \in \mathbb{R}^{N_c \times 1}$ for $t = \{1, \ldots, T\}$.

Before being passed to the learned Gaussian distribution, the encoded input is fed into a controller network, a unidirectional RNN that produces an inferred input u_t. The controller is responsible for approximating any external perturbation to the system that should be accounted for in the decoding (or generation) process. The inferred input u_t embeds the activity of external stimuli to the system and allows the model to learn system dynamics

during different global conditions (e.g., a resting versus active brain state). The accomplish this, the controller produces a posterior distribution of the inferred input conditioned on the prior state information f_{t-1} and the encoded input, represented as $P(u_t|x, g_0)\mathcal{N}(\mu_t^u, \sigma_t^u)$ where:

$$c_t = \text{RNN}^{\text{CON}}(c_{t-1}, h, f_{t-1}) \tag{3.8}$$
$$\mu_t^u = W^{\mu^u} \cdot c_t \tag{3.9}$$
$$\sigma_t^u = \exp(\frac{1}{2} W^{\sigma^u} \cdot h) \tag{3.10}$$
$$\hat{u}_t \sim \mathcal{N}(u_t|\mu_t^u, \sigma_t^u) \tag{3.11}$$
$$\tag{3.12}$$

Note that W^{μ^u} and W^{σ^u} represent learned weight matrices for the mean and variance of the inferred input distribution $P(u_t)$ and that f_{t-1} represents the decoded latent factors at the previous time step.

The controller RNN unrolls with the decoder RNN, and the inferred input $\hat{u}_t$ is then fed to the decoder (also known as the generator) to produce the latent factors at that time step f_t, representing the brain state in a lower-dimensional latent space.

$$g_t = \text{RNN}^{\text{GEN}}(g_{t-1}, \hat{u}_t) \tag{3.13}$$
$$f_t = W^{\text{Factors}} \cdot g_t \tag{3.14}$$
$$r_t = \exp(W^{\text{Rates}} \cdot f_t) \tag{3.15}$$
$$\hat{x_t} \sim \text{Poisson}(x_t|r_t) \tag{3.16}$$

Where W^{Rates} represents a learned weight matrix to map latent factors f_t to reconstructed firing rates r_t, and where $\hat{x_t}$ is the approximated reconstructed output.

The latent factors are reconstructed into firing rates for each of the N_C channels to reconstruct the input data by sampling from the posterior $P(x_t|g_0, u_{[1,\ldots,t]})$. The LFADS model is trained using self-supervised learning by minimizing the negative Poisson log-likelihood reconstruction loss and Kullback–Leibler (KL) divergence between the observed and generated activity:

$$\mathcal{L}_{LFADS} = \sum_{t=1}^{T} \log(\text{Poisson}(x_t|r_t) + D_{KL}(h\|z) \tag{3.17}$$

where D_{KL} now denotes the Kullback-Leibler divergence between the learned posterior and a Poisson prior. The latent factors f_t extracted by LFADS can capture meaningful neural dynamics such as movement intentions, decision-making states, or sensory representations, while accounting for trial-to-trial variability and external noise through the controller state.

3.5 Additional Architectures

Recent research has explored alternative architectures for decoding neural signals in addition to RNN and VAE-based architectures, most notably based on the Transformer architecture. By leveraging the transformer, these models allow for parallel computation across time and improve training efficiency and stability. Furthermore, the attention mechanisms in the Transformer have been shown to successfully model long-range dependencies in Natural Language Processing (NLP), and are also advantageous to model trial-to-trial dependencies and non-stationarities in neural data.

3.5.1 Neural Data Transformer

The Neural Data Transformer (NDT) (Ye et al. 2021) adapts the transformer architecture to model high-dimensional time-series data from neural recordings. Originally proposed for motor decoding from cortical populations, the NDT replaces the recurrence of RNNs with self-attention mechanisms that explicitly model dependencies between time steps. This allows the model to capture long-range temporal relationships more effectively than recurrent architectures, which suffer from vanishing and exploding gradients[1].

3.5.2 Spatio-Temporal Neural Data Transformer

The Spatio-Temporal Neural Data Transformer (STNDT) extends the NDT framework by encoding not only the temporal but also spatial context into the attention mechanism, making it well-suited for modeling neural population data that are simultaneously structured across time and space (Le et al. 2022). Spatial attention allows the model to weigh contributions from different neurons or channels at each timestep, enabling it to learn functional relationships within the neural population, such as coordinated firing patterns or region-specific dynamics. This architecture has shown utility in decoding complex motor behaviors and inferring latent neural representations in high-density recordings.

[1]The advantage of the Transformer architecture over Recurrent Neural Nets is highlighted in the pivotal "Attention is all you need" paper in which they were introduced (Vaswani et al. 2017).

Despite their promise, transformer-based models pose significant challenges for real-time applications such as closed-loop brain-computer interfaces. Their high memory footprint and large number of parameters currently limit their compatibility with FPGA acceleration. Key challenges include the difficulty of pruning and quantizing attention layers without significant loss in performance. However, a challenge that has been addressed is support for model layers in High-Level Synthesis for Machine Learning (HLS4ML) – as is discussed later (see chapter 4, System Implementation). Although transformer-based models were not deployed in the final decoding system presented here, their performance and deployment feasibility on FPGAs remain an active area of experimentation for future iterations of this work.

Chapter 4

System Implementation

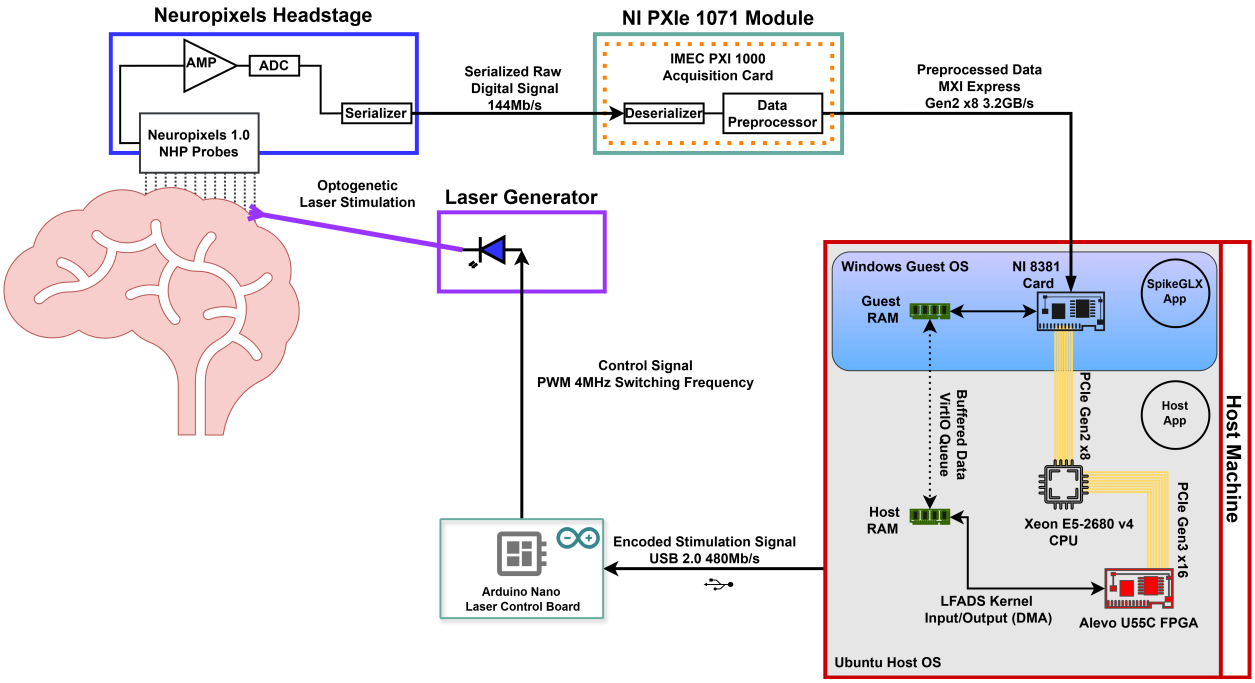

Figure 4.1: Detailed diagram of the closed-loop BCI.

This chapter introduces the crux of this work, detailing the architecture and implementation of a real-time, FPGA-accelerated Brain-Computer Interface (BCI) system for Non-Human Primate (NHP) experiments. To effectively react to brain activity in real-time with neurostimulation, the BCI must act on the order of milliseconds. For instance, to test a hypothesis on the functions of a neuron with an average firing rate of 60 Hz, the BCI must be fast enough to process incoming data and respond in $\frac{1}{60}$ seconds or 16.667ms. Specialized neurons act at significantly higher firing rates and can further constrain the total acceptable system latency. However, many CPU and GPU architectures cannot execute sophisticated

decoder models, such as deep learning based decoders, within this strict time constraint (see chapter 5, System Evaluation). The system developed in this work leverages a Field Programmable Gate Array (FPGA) to decode incoming neural data with low latency and high data throughput, orders of magnitude faster than CPUs and GPUs, enabling real-time experimentation.

The system is functionally described by three stages: data acquisition of neural activity, signal filtering and processing, and optogenetic stimulation. Altogether, these processes form a closed information loop with the brain. The decoder model is hosted as a kernel on a Xilinx U55C FPGA, continuously passing an incoming data stream through a pipelined LFADS kernel. The model was first defined in high-level Python code using the Tensorflow library. It was then iteratively compiled into RTL code to produce an FPGA firmware binary directly executable on the FPGA (for more information on the HLS4ML synthesis pipeline, refer to the HLS4ML section).

4.1 Data Acquisition

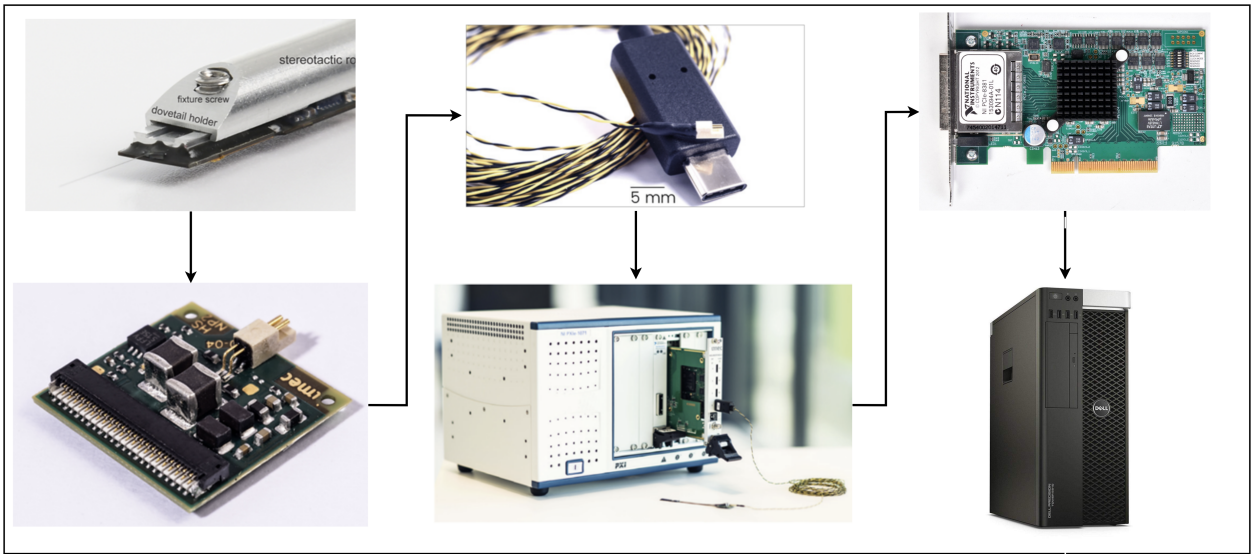

Figure 4.2: Hardware components in the Neuropixels data acquisition system, including the NHP 1.0 Probe (top left), headstage (bottom left), transmission cable (top middle), PXIe acquisition card in NI 1071 chassis (bottom middle), PXIe 8381 card (top right), and host computer (bottom right).

The data acquisition subsystem extracts raw data from the brain and produces a digital representation appropriate for the decoder model on the FPGA. Brain activity from the NHP is recorded through an Imec Neuropixels 1.0 NHP electrode array, capturing high-density

intracortical neural spiking activity. The probes offer a high spatial resolution with 960 electrode tiles tightly placed along the shank. Each probe reads data from 384 channels at a sampling frequency of 30 KHz with a mapping between a channel to a specific tile defined in the configuration file for the recording session. With a sampling frequency of 30 KHz, each channel provides a high temporal resolution of the signal, sampling roughly every 33.33μs.

The Neuropixels probes are connected to a headstage module housing an Amplifier, a Low Dropout Regulator (LDO), and an Analog-Digital Converter (ADC) which uses 10 bits to represent each sample. The digitized signal is then serialized on the headstage and transmitted to an external computer for pre-processing - an Imec PXIe Acquisition card plugged into a NI PXIe 1071 chassis. The card receives the signal through its Input/Output (I/O) ports, deserializes the signal, and generates a formatted and processed signal. The signal is outputted through the chassis backplane over a Gen2 x8 MXI Express cable to a receiving NI 8381 PCIe card in the host computer.

The host computer runs a Windows virtual Operating System (OS) to execute a data acquisition pipeline (see section On Dual Operating Systems below 4.6). The receiving 8381 card stores the data in virtual main memory as directed by the SpikeGLX application running on the Windows guest OS. The data is stored as a fixed-length buffer of length $C \cdot T$ where C and T represent the number of channels and timesteps, respectively, and are configured prior to acquisition. Next, Kernel Virtual Machine (KVM) VirtIO Queues are used by the SpikeGLX and host applications to transfer the data from the guest memory space to the host memory[1]. The host app then reshapes the data into a two-dimensional matrix $\in^{C \times T}$ as expected by the decoder architecture. Lastly, the host app uses Xilinx RunTime (XRT) drivers to transfer the 2D data to the FPGA's global memory over Direct Memory Access (DMA) via PCIe. The FPGA then pulls the data into the logic fabric over an AXI4 interface.

4.2 Hardware-Accelerated Decoding & State Estimation

Once the host transfers the input to the kernel on the FPGA, it is propagated through the decoder model to create a latent representation of the brain state. These latent factors are returned to the host machine to be used in downstream tasks specific to the experiment. For instance, the latent factors could be passed to a Fully Connected Network (FCN) to predict hand kinematics for prosthetic control.

[1]Note: The data is not actually transferred between memory locations. Instead, the guest writes to memory and passes the physical address along with a descriptor (length of data) to the KVM hypervisor. The hypervisor then passes this information to the host, which maps the address to a location in the host address space and accesses the data directly.

4.2.1 Designing the Inference Pipeline

During inference, the decoder model is provided a context window of $T = 20$ms as standard in previous BCI literature (C. et al. 2018). However, waiting 20ms to construct an input matrix of unseen activity would severely underutilize the FPGA inference throughput. Additionally, predicting the latent state at each time step $[t, t + \Delta, \ldots, t + T]$ - or in a Many-to-Many approach - would not be useful in an online setting where we are only interested in future activity (activity at $t + T + \Delta$). To address both concerns, the decoder is set up for a Many-to-One task, taking in an input context window $[t, t + T]$ and predicting activity at the future time step $t + T + \Delta$. The sliding window is realized through a First-In-First-Out (FIFO) buffer which pushes $\Delta = 1ms$ of input data into the buffer to produce each new input matrix. Note that the decoder could also perform autoregressive generation - predicting the next time-step and feeding the generated results back into the model to predict further into the future - but this could compromise the stability and accuracy of the results.

4.3 Encoding the Stimulation

The host machine uses the decoded brain state generated by LFADS to stimulate the brain using optogenetics. The BCI testbed is equipped with a laser generator controlled by an Arduino Nano Laser Control Board. The board receives control signals from the host over a USB 2.0 connection and generates a pulse-width modulation (PWM) signal to control the laser. The specific stimulation pattern is directly encoded in the signal, with a binary 1/0 turning the laser ON/OFF (the duty cycle is set to 100% during the window of stimulation and 0% otherwise). The laser is directed towards a brain site with fiber optic cables, providing coarse spatial control of the location of stimulation. The Arduino provides a fine (sub-microsecond) degree of control over the timing of stimulation with a maximum PWM Switching Frequency of 4 MHz. The primary focus in evaluating the testbed was closed-loop performance and system action latency (i.e., the time between an input signal from the brain to stimulation). To this purpose, we implemented a simple stimulation encoding by triggering the laser when the average inferred latent factors exceeded a threshold. While the study of optical stimulation techniques for unknown neural foundations is beyond the scope of this work, several insights are provided for future studies and directions (see Chapter 6, Discussion). For instance, a study aimed at identifying the behavioral impacts of a particular motor circuit might trigger an inhibitory laser stimulation when the decoder predicts that the circuit will become active, thereby isolating the circuit to evaluate changes in behavior compared to a baseline condition without optical stimulation. To this end, the testbed provides a hardware-accelerated platform

capable of investigating more interesting neuroscience questions with different decoding and stimulation methodologies.

4.4 Consideration in selecting the FPGA

When selecting an FPGA, a key consideration is the logic resource availability (LUTs, registers, DSPs, and memory) to ensure the device can accommodate both current and future workloads (e.g., extended LFADS architectures with downstream models, transformer-based models, etc). Additional factors include power consumption and an appropriate I/O bandwidth. Furthermore, the FPGA must work with a data acquisition system, so a PCIe form factor without an onboard CPU was easier to integrate with the existing workstation PC. Below are comparisons between the Alveo U55C and other viable FPGAs – namely the Alveo U50LV and the Virtex 7-series XC7VX690T, with approximate resource usage figures given LFADS resource estimates for a U55C target board.

Board	**U55C**	**U50LV**	**XC7VX690T**
Price (USD)	$4,747.00	$3,303.00	$8,094.00
PCIe	Gen4x8	Gen3x16	Gen3x4
Network Interface	2x QSFP28 100Gb	1x QSFP28 100Gb	1Gb Ethernet
LUTs (Total)	1304K	872K	693K
LUT Usage (%)	14.5%	21.7%	27.3%
Registers (Total)	2607K	1743K	866K
Reg Usage (%)	9.3%	13.9%	27.9%
Block RAM (Mb)	70.9	47.3	52.9
BRAM Usage (%)	14.9%	22.3%	19.9%
DSPs (Total)	9024	5952	3600
DSP Usage (%)	24.3%	36.8%	60.8%

Table 4.1: Comparison of FPGA Board Resource Utilization for LFADS Kernel

The U50LV has more than sufficient resources to execute the LFADS kernel, being most constrained on DSPs at 36.8% usage, and is the cheapest of the three options. However, at the time of purchase, the card was on backorder with an estimated lead time of several weeks despite surveying multiple vendors (Digikey, Colfax, Mouser, AMD). Meanwhile, the approximate lead time on the U55C was only 2 weeks, making it the next best choice given availability, cost, and compute.

4.5 HLS4ML

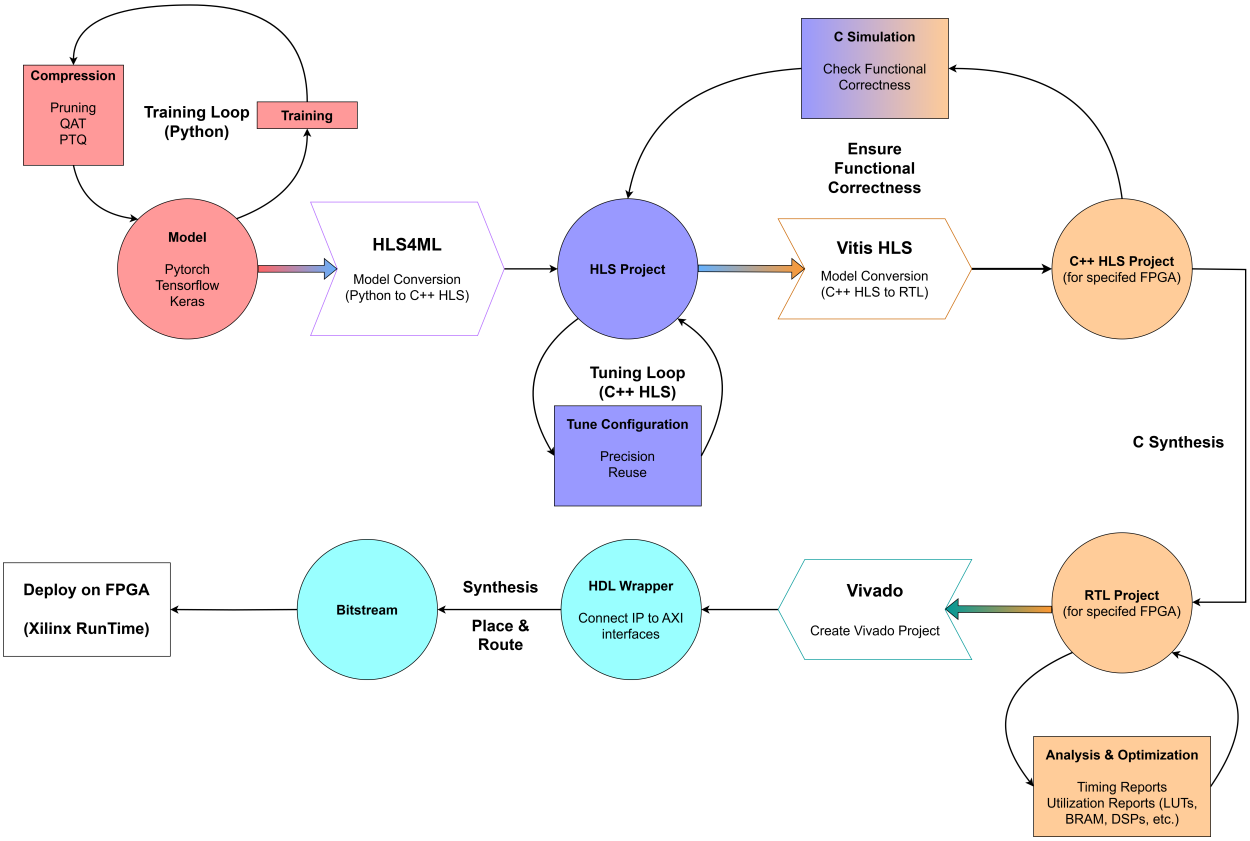

Figure 4.3: Design Flowchart of HLS4ML synthesis and testing pipeline. Note: HLS4ML supports multiple frontend and backend design flows, and this illustrates just one of many possible approaches to converting high-level Python definitions to FPGA bitstreams.

The deployment of machine learning models onto an FPGA requires the designer to bridge high-level model definitions in Python frameworks and low-level FPGA firmware. The HLS4ML framework performs this by translating neural network models, developed in Python using libraries such as TensorFlow and QKeras, into C++ code suitable for High-Level Synthesis (HLS) tools such as Xilinx's Vitis HLS (FastML Team 2024). In this workflow, a trained and quantized model is first converted into an HLS project using HLS4ML's backend conversion processes. This involves generating a configuration that specifies model parameters and desired hardware characteristics. The resulting C++ code represents the model's architecture and operations in a form useful for further hardware synthesis. The HLS code is then synthesized into Register-Transfer Level (RTL) representations in SystemVerilog using Vitis HLS. This synthesis process includes optimizations for latency, throughput, and resource utilization, guided by directives and constraints defined in the configuration. The

RTL code is then packaged into an Intellectual Property (IP) core, which can be integrated into larger FPGA designs. For deployment, the IP core is compiled into a kernel compatible with the target FPGA platform—in this case, the Xilinx Alveo U55C. This accelerator card relies on a host machine to manage data transfers, memory allocation, and kernel execution via the Xilinx RunTime (XRT) environment. Data, such as model inputs and outputs, is communicated between the host and FPGA over PCIe, enabling efficient inference operations.

4.6 On Dual Operating Systems

The system design adopts a dual operating system architecture in which a Linux (Ubuntu) host runs a Windows guest OS using the Kernel-based Virtual Machine (KVM) hypervisor. This architecture is driven by a fundamental constraint: Neuropixels data acquisition and the SpikeGLX API are only supported on Windows, while the Xilinx RunTime (XRT) drivers required to communicate with the FPGA are only available on Linux.

The Windows guest OS executes the data acquisition pipeline using the proprietary SpikeGLX software, which interfaces directly with the National Instruments 8381 PCIe card to receive Neuropixels data. Meanwhile, the Linux host manages FPGA control and memory operations through XRT. This dual-environment setup avoids the limitations of attempting to virtualize or emulate incompatible drivers (e.g. NI drivers under Linux), which could introduce instabilities in the low-latency processing required for real-time neural decoding.

To bridge the memory space between the guest and host, the system uses VirtIO queues—a high-performance shared memory communication mechanism provided by KVM. VirtIO queues allow the Windows guest to write acquired data into a buffer whose physical address is passed (along with metadata such as length) to the Linux host. The host then accesses this buffer directly, avoiding redundant memory copies. This mechanism is critical for maintaining high throughput with minimal system overhead. Alternative mechanisms, such as VirtIO serial or VirtIO socket, were also deployed in initial revisions of this system. VirtIO serial, which mimics a character device interface similar to a serial port, suffers from limited bandwidth and increased overhead due to serialization and buffering. VirtIO socket, while higher throughput than serial, is based on TCP/IP communication and requires both user-space applications to implement framing protocols and buffer management, along with introducing the overhead from constructing packets from raw data. In contrast, VirtIO queues offer direct access to guest memory, enabling seamless data transfer and tightly synchronized communication between the Windows acquisition environment and the Linux processing host.

Chapter 5

System Evaluation

Figure 5.1: The built closed-loop BCI system. Essential components are boxed for better visualization. The Host computer is in red; The IMEC data acquisition module is in green; The Laser generator is in purple; The Arduino laser control board is in blue.

The performance of the designed system is evaluated on four metrics: system latency, neural decoding accuracy under quantization, power consumption, and FPGA resource utilization. Together, these metrics provide a comprehensive understanding of the system's

capability for real-time, high-fidelity brain-computer interfacing.

5.1 Latency Characterization

Table 5.1: Latency Breakdown of the Closed-Loop BCI System

Component	**Latency (ms)**
Headstage & Module Preprocessing	2.512
Module to Host	5.047
Host to FPGA	0.018
Kernel Execution	0.156
FPGA to Host	0.008
Host to Laser Control Board	2.047
Laser Control Board to Laser Generation	0.045
Total	9.833

A fundamental constraint in any real-time Brain-Computer Interface (BCI) is system latency—the time between acquiring a neural signal and generating a corresponding output or stimulus. To characterize a fine-grained system latency, the data pipeline was discretized into components and each segment's latency was measured using a combination of software profiling and hardware timers. Then, the cumulative system latency was validated using end-to-end timing measurements across the system.

Table 5.1 summarizes the latency distribution. The average inference latency of the LFADS model across was evaluated on the three architectures, demonstrating a substantial reduction in inference latency using FPGA acceleration [1]. The most significant latency contributors were the Imec Neuropixels data acquisition module and communication overheads between the module and the host, accounting for approximately 7.59ms combined. This could not be bypassed due to the proprietary hardware and communication protocols used by the probes. Communication from the host to the stimulus control hardware via the Arduino nano control board added an additional 2.092ms.

The total system latency of 9.833**ms** demonstrates that the system is capable of responding to a wide variety of neural activity, including neurons firing at an average rate of 60 Hz (delay of 16.667ms between spikes). Moreover, real-time BCI systems are usually limited by the neural decoding latency. Notably, the neural decoding model executed on the FPGA

[1]Note that inference on the CPU is not well parallelized by the Tensorflow framework, stressing only 2 of the 16 CPU cores during testing.

demonstrated an average inference latency of only 0.156ms, comprising a marginal 1.7% of the total system latency. As shown in Table 5.2, this marks a 1650x and 118x reduction in LFADS inference latency compared to execution on a CPU and GPU, respectively. Furthermore, CPU and GPU execution of the decoder can experience large fluctuations in execution time due to varying loads on the system, with a standard deviation of 56.693ms and 2.995ms, respectively. Meanwhile, FPGA execution is far more consistent, with a standard deviation on inference latency of only 2μs, allowing far tighter tolerances on system latency. Therefore, hardware acceleration of neural decoding is highly effective in minimizing the compute latency bottleneck and enables the system to respond to brain activity in real time. While the acquisition hardware could not be bypassed or optimized in the current architecture, future system designs may consider custom acquisition pipelines to further reduce end-to-end latency.

5.2 Power Utilization

An additional key criterion in evaluating the BCI system is the system power consumption - a critical design consideration for developing future portable and long-term usage BCI devices. The average power usage of the FPGA was compared to a CPU and GPU while executing inference of the LFADS model across 10000 data samples. The power consumption of the FPGA was measured as the total system power consumption reported by XRT driver tools; the CPU power was measured as the CPU package power; and the GPU power was measured as the GPU core input power. Both CPU and GPU power figures were reported by HWiNFO, a tool that monitors onboard power, thermal, and frequency sensors through hardware drivers. To isolate the contribution of the inference algorithm to the power consumption and account for background processes, the active power draw for all three architectures was derived by subtracting their respective idle and runtime power consumptions. As shown in Table 5.3, average power consumption on the FPGA was 27.867 W, utilizing only 33.03% and 31.89% of the power compared to the CPU (84.345W) and GPU (87.393 W), respectively. The results highlight both the computational performance and power efficiency gained by accelerating the BCI system with an FPGA.

Table 5.2: Inference Latency & Power Across Architectures

Architecture	**Inference Latency (ms)**	**Active Power Draw (W)**
FPGA	0.156 ± 0.002	12.447
CPU[†]	257.516 ± 56.693	39.490
GPU[‡]	18.551 ± 2.995	52.606

[†] AMD 5950x 16-Core CPU
[‡] NVIDIA RTX 3090 GPU

5.3 Decoding Accuracy

Multiple variants of the LFADS architecture were trained and deployed with different bit-resolutions for the model weights and a varying latent factor dimensionality. The model was evaluated on its reconstruction loss (Negative Log-Likelihood, R^2 score) across different architectures. The final model used 10 latent factors with 8-bit quantization of the weights to balance its resource utilization with reconstruction performance. The final architecture was then additionally evaluated on the Hand Reach-Out task, classifying the direction of hand movement of a Non-Human Primate (NHP) in a controlled experiment.

5.3.1 Quantization

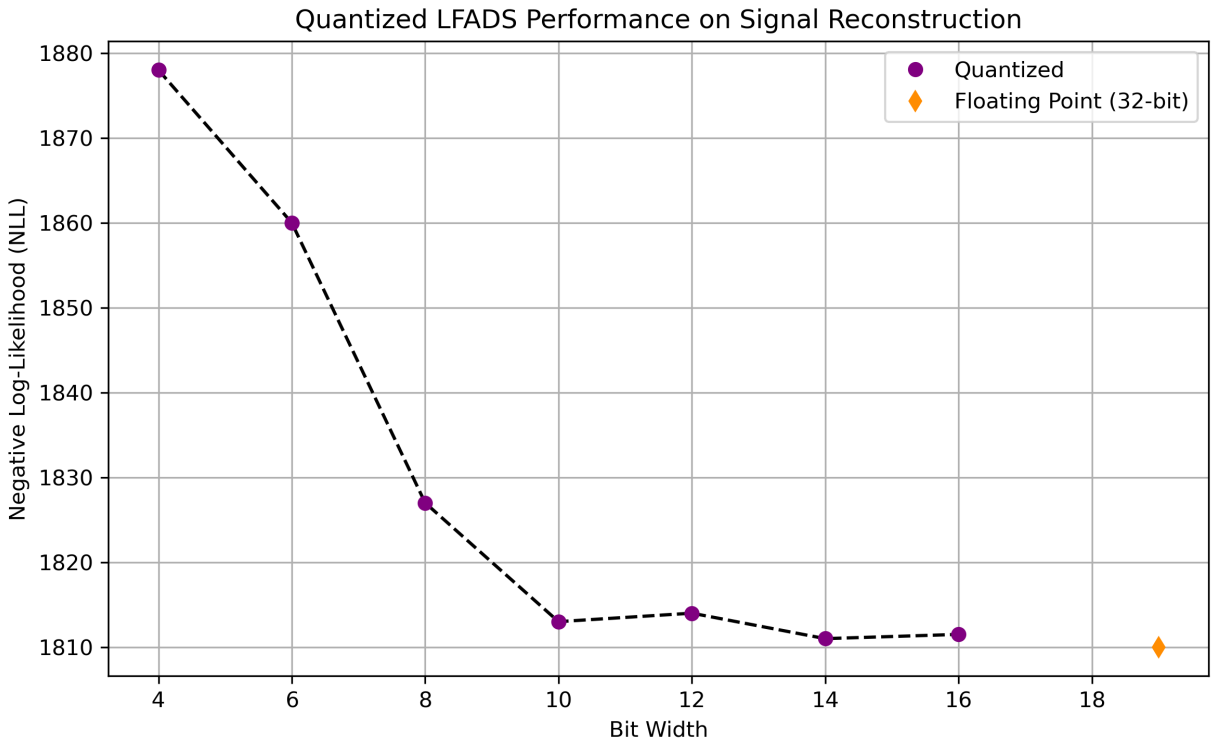

Figure 5.2: Effects of the degree of quantization on the reconstruction loss of LFADS (lower is better).

To reduce the resource utilization of the LFADS kernel on the FPGA logic fabric, the model was quantized to use 8-bit integers for weights and activations as opposed to the standard 32-bit floating point representation. This reduced the reconstruction accuracy of the model, as shown in the table below.

Table 5.3: Bit Precision & Decoding Accuracy

Bit Precision	**Decoding Accuracy**[†]
32-bit float (Full-precision)	61.18%
8-bit int (Quantized)	55.21%

[†] Model evaluated on movement direction classification task for the Hand Reach-Out experiment

The effects of the degree of quantization on the LFADS model were tested with the reconstruction loss, or the Negative Log Likelihood, assuming a Poission generative model. While quantization significantly reduces computational complexity and memory bandwidth, it can compromise model accuracy, particularly for architectures sensitive to weight precision. The reconstruction accuracy significantly deteriorates under extreme quantization (4-bit, 6-bit representations of weights) while higher bit-widths (14-bit, 16-bit) offer near identical performance to the floating point model. However, the 8-bit model was chosen for deployment on the FPGA due to its balance between performance and resource utilization.

5.3.2 Latent Factors Dimensionality

Another key design choice in tuning the performance of the LFADS decoder was the dimensionality of the latent space approximated by the model. To test this, the model was trained on data collected on the hand-reach out task at the aoLab at the University of Washington. The task involves an NHP implanted with Neuropixels intracortical probes performing multiple trials of the reach-out experiment, where it must move its hand to one of eight target circles when a cue is provided. The decoder model classifies the movement direction of the hand given the raw neural data. As seen in Figure 5.2, the LFADS decoder achieves its best test accuracy of 62.84% with 22 latent factors. However, it is not consistently true that a higher latent dimensionality improves performance, as shown by the steep decline in accuracy when shifting from 18 to 20 latent factors. Furthermore, increasing latent factor dimensionality will also yield a model with more parameters, as the weight matrix W^{Factors} scales proportionally with the number of latent factors. Therefore, increasing the latent factors creates a tradeoff in the model accuracy and its resource utilization on the FPGA. To strike a balance between the two, a latent dimensionality of 10 factors was used in the final LFADS model deployed onto the FPGA.

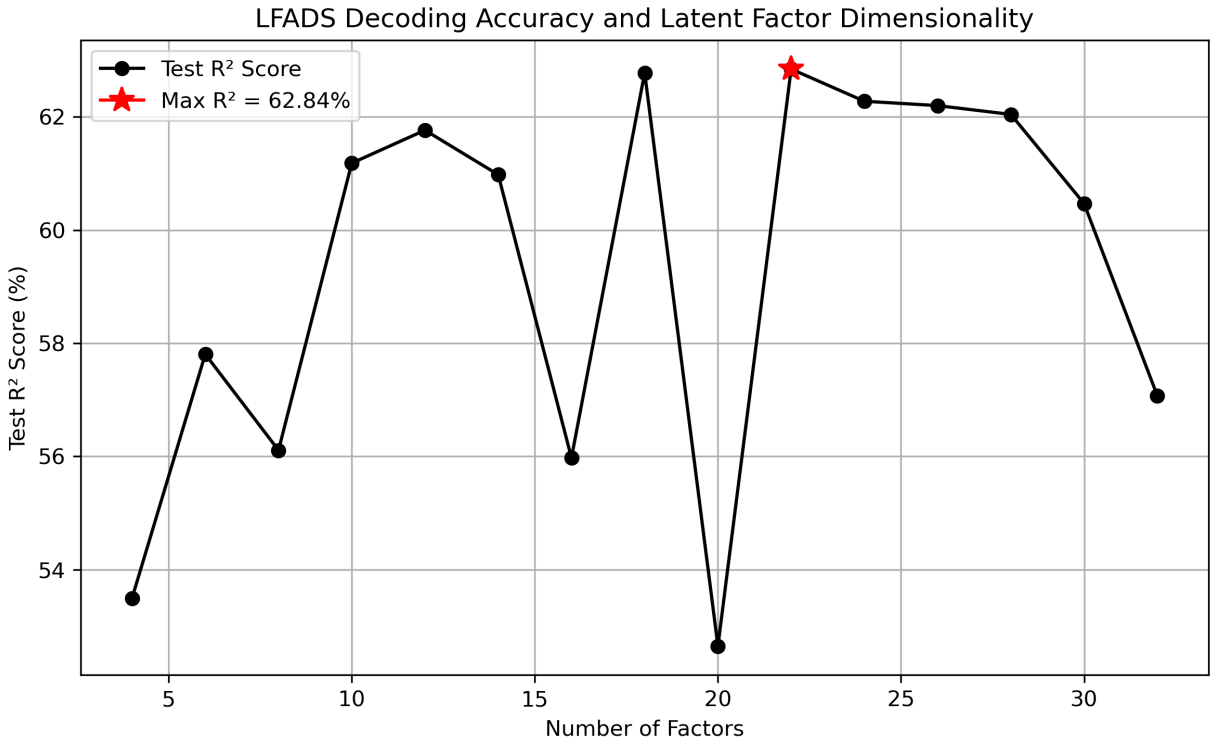

Figure 5.3: Effects of latent space dimensionality on signal reconstruction for LFADS (R^2 score, higher is better).

5.3.3 The Hand Reach-Out Task

The final 8-bit 10-factor LFADS model was trained and tested on data collected from a Hand Reach-Out experiment conducted at the aoLab at the University of Washington. The experiment consists of an NHP subject moving its hand from a center circle to one of eight peripheral circles distributed at a radial distance of 6.5cm away from the center. To begin a trial, the primate is required to hold its hand over the center circle for a short time (variable to the experiment, between $100 - 600$ms). Then, it is provided with a go-cue to a highlighted target circle. For the trial to be considered successful, the NHP had to reach the target circle within 3s and hold at the target circle for an additional 200ms. The true position and movements of the primate's arm were recorded using Optitrack motion capture cameras (OptiTrack 2025). Neural recordings were acquired using the same Neuropixels 1.0 NHP probes deployed in the system, with data from 384 channels sampled at 30 kHz. Spiking activity was detected by thresholding the signal at five standard deviations (5σ) above the signal's root-mean-square (RMS) value for each trial.

Successful trials were rewarded with juice while failed trials (e.g., early movement, timeout, or target miss) were repeated, encouraging the primate to learn the task and provide a

consistent best effort across trials. The movement directions were sampled uniformly and, across multiple trials, provided well-represented class distributions across movement directions for robust neural population analysis.

To perform hand movement classification, a smaller Multi-Layer Perceptron (MLP) model with two hidden layers was appended to the LFADS model. The MLP takes as input the latent factor representation of the neural data and produces probability estimates across eight classes, one for each movement direction. In this extended architecture, the LFADS model acts as a feature extraction pipeline to transform the neural data into low-dimensional representations of the brain state. A downstream model, in this case the MLP head, uses the features to perform regression and classification tasks.

The MLP head was also quantized to 8 bits for deployment on the FPGA. The **model achieved a classification accuracy of** 73.95% on the test set.

5.4 FPGA Resource Utilization

Resource efficiency is an important metric in evaluating the scalability of the decoding algorithm for different experiments, especially since the decoder produces an estimate of the brain state that can be used for downstream classification and regression models. The 8-bit quantized LFADS model with 12 latent factors was synthesized and deployed on the Xilinx U55C FPGA.

Table 5.4: LFADS Kernel Resource Utilization on Xilinx Alveo U55C

Resource	**Utilization**	**Available**	**Utilization (%)**
LUTs	181,256	1,304,000	13.90%
Flip-Flops	160,331	2,607,000	6.15%
Block RAM	4.28 Mb	70.90 Mb	6.03%
DSP Blocks	1665	9024	18.44%

As seen in Table 5.4, the deployed model occupied only a modest fraction of the available logic, memory, and compute resources. Notably, the use of BRAM and DSPs remains well below saturation, leaving ample headroom for further architectural exploration, such as:

- Deployment of more complex models (e.g, transformer-based NDT and STNDT)
- Parallel decoding pipelines for multi-region neural recordings
- Integration of additional downstream models (e.g., classification head for the hand-reach out task)

Chapter 6

Discussion & Future Works

In this work, we present an end-to-end design of an integrated Brain-Computer Interface (BCI) system, combining high-throughput neural data acquisition, low-latency processing, and hardware-accelerated decoding into a unified architecture. The proposed system leverages both software and hardware components: real-time neural data is acquired via Neuropixels probes and SpikeGLX in a Windows virtual machine, while the data is processed through a quantized deep learning model on a Xilinx FPGA hosted by a Linux environment. The dual-operating system design enables seamless integration of otherwise incompatible toolchains, and our use of VirtIO queues ensures efficient, zero-copy data transfer between environments, thereby minimizing latency and maximizing throughput.

To the best of our knowledge, this work represents one of the first complete implementations of a fully integrated closed-loop BCI system built from the ground up with hardware-software co-design principles. By targeting not only the fidelity of the neural decoding model but also the physical constraints of deployment, such as inter-OS communication, PCIe bandwidth utilization, and quantized inference, we take a step toward bridging the gap between theoretical BCI pipelines and their practical realization in clinical or mobile settings.

We demonstrated that our system is capable of reliably transferring and decoding high-density neural data in real-time, while meeting critical constraints on power, latency, and memory. Our FPGA implementation of a neural decoder, quantized to 8-bit integer precision, achieves high decoding accuracy comparable to its full-precision software counterpart. Moreover, the reduced power consumption and deterministic execution latency of the FPGA design make it suitable for mobile or implantable platforms—an essential property for next-generation BCIs.

Despite its contributions, our work leaves several open avenues for exploration. One notable limitation is the exclusive use of Neuropixels 1.0 probes for neural recording. While Neuropixels provides dense, single-unit resolution signals ideal for testing decoding architectures, its use

is often constrained to laboratory settings due to cost, setup complexity, and invasiveness. Broader adoption of BCI technologies will require compatibility with more accessible recording modalities such as EEG, EMG, or ECoG. Future extensions of this framework could investigate adapting our system to interface with these alternative modalities, evaluating how the change in signal quality and structure affects decoder design and hardware requirements. Another area for improvement lies in further optimizing system latency in other stages of the BCI pipeline. In the current implementation, control signals are driven via an Arduino Nano board over USB 2.0. The latency of transmitting and processing the control signal could be reduced by replacing Arduino with faster microcontroller platforms and communication protocols (e.g., a simple control signal using GPIO on a Raspberry Pi).

Additionally, while we have demonstrated a unidirectional pipeline—from recording to decoding—the full realization of a closed-loop BCI requires the ability to stimulate the brain based on decoded outputs. To that end, we have outlined the use of optogenetics as a candidate stimulation technique. The promise of optogenetic control lies in its cell-type specificity and high temporal resolution; however, its implementation requires complex experimental protocols, including viral vector delivery, light delivery systems, and precise timing control. Although the integration of optogenetics was beyond the scope of this study, future work will explore its application in shaping or perturbing neural dynamics as part of closed-loop feedback systems. In particular, this could open pathways to therapeutically relevant applications such as seizure interruption, motor rehabilitation, or cognitive enhancement.

From a model design perspective, our implementation relied on LFADS (Latent Factor Analysis via Dynamical Systems) as the decoding architecture, chosen for its structured temporal modeling and interpretability. However, the landscape of neural signal modeling has rapidly evolved, and transformer-based architectures—notably those adapted for continuous or non-stationary data—are emerging as state-of-the-art for sequence modeling. Recent works such as STNDT (Spatio-Temporal Neural Data Transformers) have demonstrated the ability to model complex neural trajectories across time and space. Exploring the feasibility of deploying such architectures on FPGAs, particularly under quantized constraints, is a promising future direction. Challenges will include the memory footprint of self-attention mechanisms, hardware-efficient attention approximations, and scheduling multi-layered transformer blocks under strict latency budgets.

Finally, the current system demonstrates co-design at the algorithmic and systems level but remains in a proof-of-concept stage. Future work will focus on scaling the platform for long-term, real-world use. This includes supporting persistent streaming, improving decoder generalizability across sessions and subjects, and incorporating adaptive learning strategies to compensate for neural signal drift. Furthermore, evaluating the system's performance in

in vivo or behavioral loop settings will be essential to validate its translational potential.

In summary, this work introduces a robust and flexible framework for high-speed, low-power neural decoding via tightly coupled software and hardware components. Through careful system design, we enable a real-time neural interface that is both high-performing and resource-aware. The insights gained from this implementation lay a foundation for future BCIs that are more adaptive, scalable, and accessible, moving one step closer to practical neural interfaces capable of operating beyond the lab.

Chapter 7

Individual Contributions

This work is the product of the efforts of multiple students across different domains. Development of the HLS4ML integration of the LFADS model, including creating custom layer definitions in C++ HLS from the Tensorflow model, was conducted by my predecessor, Xiohan Liu, and collaborators at the National Yang Ming Chiao Tung University in Hsin-Chu, Taiwan - namely ChiRui Chen and YanLun Huang. After I joined, I worked closely with ChiJui Chen to synthesize, tune, and troubleshoot the LFADS deployment.

The neural data acquisition was set up and running at the aoLab, though it was not yet configured for online experimentation. I was responsible for integrating the FPGA system with the existing experimental apparatus, including writing software to handle the data passing, running closed-loop tests, and characterizing the system's accuracy, power, and latency figures. In this integration step, I received help from Leo Scholl and Ryan Canfield, members of the aoLab, to work with the SpikeGLX application interface and gain access to previously collected data to test the LFADS model.

After the system had been integrated, validated by passing noise data from the probes to the host machine and triggering the laser to close the loop, I began working with Hao Fang to test the LFADS model on the Hand Reach Out task. This involved extending the LFADS model with an MLP classification head and organizing the data for training. Hao Fang developed starter code for the extended model and a data loader to work with previously collected data at the aoLab. ChiRui Chen then used this code to test variants of the LFADS model on the task and report the accuracy. I took the trained model and synthesized it to the U55C FPGA in the lab to validate the simulated resource utilization and timing reports.

Though not directly involved in the deployment of the system, Professors Scott Hauck, Amy Orsborn, and Shih-Chieh Hsu have been instrumental in guiding the direction of this effort. Additionally, Trung Le and Leo Scholl helped to decide the appropriate data acquisition system for our work and identify the constraints placed on the system for various experiments.

References

Boyden, E., F. Zhang, and e. a. Bamberg E. (2005). "Millisecond-timescale, genetically targeted optical control of neural activity". In: *Nature Neuroscience* 8, pp. 1263–1268.

C., P., D. O'Shea, and e. a. Collins J. (2018). "Inferring single-trial neural population dynamics using sequential auto-encoders". In: *Nature Methods* 15, pp. 805–815.

FastML Team (2024). *fastmachinelearning/hls4ml*. Version v1.0.0. DOI: 10.5281/zenodo.1201549. URL: https://github.com/fastmachinelearning/hls4ml.

Hodgkin, A. and A. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117, pp. 500–544.

Le, T. and E. Shlizerman (2022). "STNDT: Modeling Neural Population Activity with a Spatiotemporal Transformer". In: *NIPS*.

Lodato, S. and P. Arlotta (2015). "Generating neuronal diversity in the mammalian cerebral cortex". In: *Annual Review of Cell and Developmental Biology* 31, pp. 43–66.

Malik W.Q., e. a. (2010). "Efficient Decoding With Steady-State Kalman Filter in Neural Interface Systems". In: *IEEE Transactions on Neural Systems Rehabilitation and Engineering*, pp. 25–34.

Minagar, A., J. Ragheb, and R. Kelley (2003). "Smith surgical papyrus: description and analysis of the earliest case of aphasia". In: *J Med Biogr.* 11 (2).

OptiTrack (2025). *optitrack/motioncapture*. URL: https://optitrack.com/cameras/.

Pascanu R., e. a. (2013). "On the difficulty of training recurrent neural networks". In: *ICML* 28.

Sussillo, D., J. Jozefowicz, L. Abbott, and C. Pandarinath (2016). "LFADS - Latent Factors Analysis via Dynamical Systems". In: *arXiv*. DOI: https://doi.org/10.48550/arXiv.1608.06315.

Tampuu A., e. a. (2019). "Efficient neural decoding of self-location with a deep recurrent network". In: *PLOS Computational Biology*.

Vaswani, A., N. Shazeer, N. Parmar, and e. a. Uszkoreit J. (2017). "Attention Is All You Need". In: *Neural Information Processing Systems*.

Vilela, M. and L. Hochberg (2020). "Applications of brain-computer interfaces to the control of robotic and prosthetic arms". In: *Handbook of Clinical Neurology* 168, pp. 87–99.

Ye, J. and C. Pandarinath (2021). "Representation learning for neural population activity with Neural Data Transformers". In: *arXiv*. DOI: https://doi.org/10.51628/001c.27358.