VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

COURSE'S ASSIGNMENT REPORT

AIT3012: COMPUTATIONAL NEUROSCIENCE AND APPLICATIONS

Topic

Exploring Neuron Models and Neural Networks

Supervisor: Professor Nguyen Linh Trung

Student:	Tran Xuan Bao
ID:	23020332
Class:	K68-AI2

Ha Noi - 2025

ACKNOWLEDGMENTS

First and foremost, I would like to express my most sincere gratitude to my supervisors and lecturers, **Professor Nguyen Linh Trung** and **Professor Guy Nagels**.

Throughout the duration of the course **AIT3012: Computational Neuroscience and Applications**, I have received exceptional support, professional mentorship, and valuable insights from them. Their deep knowledge of the biophysical mechanisms of the brain and their guidance on neural modeling have been instrumental in helping me bridge the gap between biological theory and computational simulation.

I would also like to extend my heartfelt thanks to all the **Teaching Assistants** of the course. Their constant availability, technical assistance during the programming sessions, and constructive feedback were crucial in enabling me to successfully implement the complex Python simulations required for this report.

In the process of finalizing this report, while I have strived for professional standards, there may be areas for further optimization. Therefore, I respectfully look forward to receiving your valuable comments and feedback so that my work can be further improved and perfected.

I would like to express my sincere thanks!

Hanoi, December 29, 2025

OVERVIEW

This report outlines the theoretical investigation, mathematical formulation, and programming implementation conducted for the assignment on Exploring Neuron Models and Neural Networks. The primary objective was to assess the computational properties of biological neurons and the capabilities of brain-inspired network architectures.

The study begins with a rigorous analysis of single-neuron dynamics. It contrasts the biophysically detailed **Hodgkin-Huxley (HH)** model, which describes the non-linear conductance of ion channels, with the phenomenological **Leaky Integrate-and-Fire (LIF)** model, which offers computational efficiency for large-scale networks. The report also explores the theoretical underpinnings of synaptic plasticity via Hebbian Learning.

Moving beyond single neurons, the report investigates advanced architectures including **Spiking Neural Networks (SNNs)** and **Reservoir Computing (RC)**. Specifically, an Echo State Network (ESN) was implemented to demonstrate the ”fading memory” property required for processing temporal data.

The practical component of this project involved developing a modular Python simulation framework. This framework successfully visualized the generation of Action Potentials in the HH model, the sub-threshold integration in the LIF model, and the accurate prediction of the chaotic Mackey-Glass time series using Reservoir Computing. These results highlight the trade-offs between biological realism and computational cost, providing a foundation for future work in neuromorphic engineering.

Contents

1 Theoretical Investigation of Neuron Models

1.1 The Hodgkin-Huxley (HH) Model

1.1.1 Historical Context

The Hodgkin-Huxley (HH) model is considered one of the most significant achievements in computational neuroscience. Developed in 1952 by Alan Hodgkin and Andrew Huxley based on experiments performed on the giant axon of the Atlantic squid, this model provided the first quantitative description of the propagation of action potentials. Their work, which earned them the Nobel Prize in Physiology or Medicine in 1963, demonstrated that membrane potential changes are caused by changes in the permeability of the cell membrane to specific ions, a fundamental concept extensively detailed in [1].

1.1.2 Physical Intuition

The model relies on an electrical circuit analogy for the cell membrane. The lipid bilayer acts as a capacitor (C_m), separating charge between the intracellular and extracellular environments. Embedded within this bilayer are ion channels that act as variable resistors (conductances). The electrochemical gradients driving the ions act as batteries (E_{ion}). The total current flowing across the membrane is the sum of the capacitive current and the ionic currents flowing through Sodium (Na^+), Potassium (K^+), and Leakage (L) channels.

1.1.3 Mathematical Formulation

According to Kirchhoff's current law, the evolution of the membrane potential V is described by the following differential equation:

$$C_m \frac{dV}{dt} = I_{ext} - I_{Na} - I_K - I_L \tag{1}$$

Substituting the conductance formulations, the full equation becomes:

$$C_m \frac{dV}{dt} = I_{ext} - \bar{g}_{Na} m^3 h (V - E_{Na}) - \bar{g}_K n^4 (V - E_K) - \bar{g}_L (V - E_L) \tag{2}$$

Here, $\bar{g}_x$ represents the maximum possible conductance for ion channel x, and E_x is the Nernst reversal potential. The terms m, h, and n are dimensionless gating variables between 0 and 1, representing the probability of ion channel gates being open. Their dynamics are governed by first-order ordinary differential equations [1]:

$$\frac{dx}{dt} = \alpha_x(V)(1 - x) - \beta_x(V) x, \quad \text{for } x \in \{m, h, n\} \tag{3}$$

Where $\alpha_x(V)$ and $\beta_x(V)$ are voltage-dependent rate constants derived empirically by Hodgkin and Huxley.

1.1.4 Explanation of Components

The power terms associated with the gating variables reflect the physical structure of the channels:

- **Sodium Current (I_{Na}):** Governed by m^3h. The m^3 term suggests the activation gate consists of three subunits that must all be open for conduction. The h term represents a single inactivation particle that blocks the channel at high voltages.
- **Potassium Current (I_K):** Governed by n^4. This models the ”delayed rectifier” behavior, requiring four activation subunits to open simultaneously. K^+ channels in this model do not inactivate.
- **Leakage Current (I_L):** A passive current (mostly Chloride ions) with constant conductance $\bar{g}_L$.

1.1.5 Mechanism of Action

The HH model accurately simulates the action potential through the interplay of positive and negative feedback loops:

1. **Depolarization:** When I_{ext} raises V above a threshold, voltage-gated Na^+ channels open (m increases rapidly). This causes an influx of Na^+, further depolarizing the cell (positive feedback).
2. **Peak and Repolarization:** As V approaches E_{Na}, two things happen: the Na^+ channels inactivate (h drops toward 0), and the slower K^+ channels open (n increases). K^+ flows out of the cell, opposing the depolarization.
3. **Hyperpolarization:** The K^+ channels close slowly, causing the membrane potential to briefly dip below the resting potential (undershoot) before returning to equilibrium.

1.1.6 Modern Usage

While computationally expensive due to the need to solve four coupled differential equations per neuron, the HH model remains the ”gold standard” for biophysical accuracy. It is widely used today in detailed single-neuron simulations (e.g., in the NEURON software environment) and in studies where the specific dynamics of ion channel drugs or genetic channelopathies are being investigated [2].

1.2 The Leaky Integrate-and-Fire (LIF) Model

1.2.1 Historical Context and Concept

While the Hodgkin-Huxley model offers biophysical precision, it is computationally heavy. The Leaky Integrate-and-Fire (LIF) model, which actually pre-dates HH (proposed by Louis Lapicque in 1907), takes a phenomenological approach. It abstracts away the complex ion channel dynamics, focusing solely on the neuron's sub-threshold membrane potential and the timing of output spikes. It treats the neuron not as a complex biological entity, but as a simple electrical circuit that "integrates" inputs and "fires" when a limit is reached.

1.2.2 Physical Intuition

The LIF model represents the neuron as a parallel RC circuit consisting of a capacitor (the membrane) and a resistor (the leakage channel).

- **Integrate:** The capacitor accumulates charge from input currents, representing the summation of postsynaptic potentials.
- **Leak:** The resistor allows charge to slowly dissipate, representing the constant diffusion of ions trying to return the cell to its resting equilibrium.
- **Fire:** When the potential hits a specific threshold, a spike is declared, and the system is forcibly reset.

1.2.3 Mathematical Formulation

The sub-threshold dynamics are governed by standard circuit theory. By applying Kirchhoff's law and dividing by the leak conductance g_L, we obtain the standard LIF differential equation as derived in [2]:

$$\tau_m \frac{dV}{dt} = -(V - E_L) + R_m I(t) \tag{4}$$

Here, $\tau_m = R_m C_m$ is the membrane time constant, which dictates how fast the neuron forgets its past inputs. E_L is the resting potential.

Unlike the HH model, the spike generation is not contained within the differential equation. It is an explicit algorithmic rule added on top:

$$\text{if } V(t) \geq V_{th} \quad \text{then} \quad \begin{cases} V(t) \leftarrow V_{reset} \\ \text{emit spike at time } t \end{cases} \tag{5}$$

To simulate the absolute refractory period, the potential can be clamped at V_{reset} for a duration Δ_{ref} after a spike.

1.2.4 Comparison with the Hodgkin-Huxley Model

The choice between HH and LIF represents the fundamental trade-off in computational neuroscience between biophysical realism and computational efficiency.

- **Biophysical Realism:**
  - **HH:** High. It explicitly models channel gating kinetics, correctly reconstructing the shape of the action potential and non-linear effects like threshold adaptation.
  - **LIF:** Low. It treats the spike as a point event (Dirac delta function) with no physical shape. It ignores the specific ionic mechanisms (Na/K dynamics).
- **Dimensionality and Complexity:**
  - **HH:** 4-dimensional (V, m, n, h). Requires solving four coupled non-linear differential equations. Computationally expensive.
  - **LIF:** 1-dimensional (V). Requires solving one linear differential equation. Computationally very cheap.
- **Applications:**
  - **HH:** Used for single-neuron dynamics, pharmacology, and explaining how spikes are generated.
  - **LIF:** Used for network dynamics, learning rules, and large-scale simulations involving thousands or millions of neurons [3].

1.2.5 Modern Usage

Due to its mathematical simplicity, the LIF model is the standard building block for Spiking Neural Networks (SNNs) in machine learning and neuromorphic engineering (e.g., Intel's Loihi chip). It captures the essential "all-or-none" property of neural communication without the heavy computational overhead of ion channel modeling.

1.3 The Hebbian Learning Algorithm

1.3.1 Historical Context

While the Hodgkin-Huxley and LIF models describe the immediate electrical behavior of neurons, they do not explain how the brain learns. In 1949, Donald Hebb published *The Organization of Behavior*, postulating a mechanism for synaptic plasticity. His famous maxim, often paraphrased as ”cells that fire together, wire together,” suggests that if a presynaptic neuron consistently participates in firing a postsynaptic neuron, the connection between them is strengthened [2].

1.3.2 Physical Intuition

Biologically, this corresponds to Long-Term Potentiation (LTP). When a synapse is successfully activated, biochemical processes (often involving Calcium influx and NMDA receptors) increase the efficacy of that synapse. This allows the network to store associations: if input A and input B frequently occur simultaneously, the neurons representing them form strong links, creating an associative memory.

1.3.3 Mathematical Formulation

In artificial neural networks, this biological principle is formalized as a local learning rule. The change in synaptic weight Δw_{ij} between a presynaptic neuron j and a postsynaptic neuron i is proportional to the correlation of their activities:

$$\Delta w_{ij} = \eta \cdot y_i \cdot x_j \tag{6}$$

Where:

- η is the learning rate (a small positive constant).
- y_i is the output activity of the postsynaptic neuron.
- x_j is the input activity from the presynaptic neuron.

Stability Issues: A major theoretical limitation of basic Hebbian learning is positive feedback; weights tend to grow to infinity. To address this, normalization terms are often added, such as in **Oja’s Rule**, which includes a decay term to constrain the weights:

$$\Delta w_{ij} = \eta(y_i x_j - \alpha y_i^2 w_{ij}) \tag{7}$$

1.3.4 Comparison with HH and LIF Models

It is crucial to distinguish the role of Hebbian Learning from the neuron models discussed in sections 1.1 and 1.2.

- **Dynamics vs. Plasticity:**
  - **HH and LIF:** These model the *neural dynamics* (fast timescale, milliseconds). They define how the state variable V (membrane potential) changes in response to input I.
  - **Hebbian Learning:** This models *synaptic plasticity* (slow timescale, seconds to days). It defines how the system parameters w (weights) change in response to the correlation of activities.
- **Integration in Simulations:**
  - A simulation using only HH or LIF is static; it reacts to inputs but does not "learn."
  - A simulation using only Hebbian equations has no activity to drive the learning.
  - **Synergy:** Modern simulations typically use LIF neurons to generate spikes, and a Hebbian-derived rule (like Spike-Timing-Dependent Plasticity, STDP) to adjust the weights based on the precise timing of those spikes [3].

1.3.5 Modern Usage

Hebbian learning remains the foundation of unsupervised learning in neural networks. In the context of Spiking Neural Networks (SNNs), the Hebbian principle has evolved into **STDP** (Spike-Timing-Dependent Plasticity), where the precise temporal order of spikes determines whether a synapse is strengthened (LTP) or weakened (LTD).

2 Theoretical Investigation of Spiking Networks

2.1 Spiking Neural Networks (SNNs)

2.1.1 Concept and Evolution

Spiking Neural Networks (SNNs) are often referred to as the ”third generation” of neural networks, following Perceptrons and standard Deep Learning models [3]. Unlike previous generations which use continuous valued activation functions (like Sigmoid or ReLU), SNNs operate using discrete events called spikes. This architecture bridges the gap between neuroscience and machine learning, aiming to replicate the massive parallelism and energy efficiency of the biological brain.

2.1.2 Architecture and Encoding

An SNN is typically composed of Leaky Integrate-and-Fire (LIF) neurons (as defined in Section 1.2). The critical difference lies in information encoding. In an ANN, a value of 0.8 represents high activation. In an SNN, this ”0.8” must be encoded into the time domain:

- **Rate Coding:** The information is contained in the firing frequency (number of spikes per second).
- **Temporal Coding:** The information is encoded in the precise timing of spikes (Time-to-First-Spike) or the relative latency between spikes.

Mathematically, the input to a neuron i is not a static value, but a sum of Dirac delta functions arriving at specific times t_k:

$$I_i(t) = \sum_j w_{ij} \sum_k \delta(t - t_{j,k}) \tag{8}$$

2.1.3 Training Challenges

Training SNNs is notoriously difficult because spikes are non-differentiable operations (a step function has zero derivative everywhere except at the threshold, where it is undefined). Standard Backpropagation cannot be applied directly. Modern solutions include:

- **Shadow/Surrogate Gradients:** Approximating the spike derivative as a smooth function during the backward pass [3].
- **STDP:** Using unsupervised Hebbian-based rules (Section 1.3) to locally adapt weights.

2.1.4 Comparison with Standard ANNs

- **Efficiency:** ANNs are energy-intensive as every neuron computes every cycle. SNNs are *event-driven*; computation only occurs when a spike happens. On neuromorphic hardware (e.g., Intel Loihi), this results in orders of magnitude lower power consumption.
- **Accuracy:** SNNs generally trail behind ANNs in pure accuracy on static datasets (like ImageNet) but excel in processing dynamic, temporal data from event cameras or sensors.

2.2 Reservoir Neural Networks (RNNs)

2.2.1 Concept and Motivation

Recurrent Neural Networks (RNNs) are powerful for time-series data but are difficult to train due to the "vanishing gradient" problem. Reservoir Computing (RC), exemplified by the **Echo State Network (ESN)**, proposes a radical simplification: instead of training the entire recurrent loop, we only train the final output layer. This approach is thoroughly reviewed by Lukoševičius and Jaeger in [4].

2.2.2 Architecture

An ESN consists of three distinct parts:

1. **Input Layer (W_{in}):** Maps the input signal into the reservoir. These weights are random and *fixed*.
2. **The Reservoir (W_{res}):** A large, sparsely connected network of neurons (often non-linear analog units, though spiking reservoirs exist). These recurrent weights are random and *fixed*.
3. **Readout Layer (W_{out}):** Linear connections mapping the reservoir state to the output. These are the *only trainable weights*.

2.2.3 Mathematical Dynamics

The state of the reservoir $x(t)$ evolves according to:

$$x(t+1) = f(W_{res} \cdot x(t) + W_{in} \cdot u(t+1)) \tag{9}$$

where f is usually the hyperbolic tangent (tanh). The output is simply:

$$y(t+1) = W_{out} \cdot x(t+1) \tag{10}$$

2.2.4 The Echo State Property

For the system to work, the reservoir must possess the "Echo State Property." This means the reservoir acts as a fading memory of the input history. Mathematically, this requires the spectral radius (largest absolute eigenvalue) of W_{res} to be less than 1 ($\rho(W_{res}) < 1$). If $\rho \geq 1$, the network may become chaotic or unstable [4].

2.2.5 Comparison with Deep Learning (BPTT)

- **Speed:** Training an ESN is extremely fast because it is a simple linear regression problem (solving $Y = W_{out}X$), which has a closed-form solution. It avoids the iterative, slow convergence of Backpropagation Through Time (BPTT).
- **Complexity:** Standard RNNs/LSTMs require optimizing thousands of recurrent parameters. ESNs bypass this, effectively treating the recurrent dynamics as a fixed non-linear projection kernel.

3 Programming Simulation

To ensure robustness and modularity, the simulations were implemented using a professional Object-Oriented Programming (OOP) structure in Python. The project was organized into a `models` package containing the logic for each network type, a `plotting` package for scientific visualization, and a `main.py` entry point.

3.1 Hodgkin-Huxley Model Simulation

We simulated the biophysical dynamics of a neuron under a prolonged current injection of $20\mu A/cm^2$ lasting from $t = 10ms$ to $t = 110ms$. The results are visualized in the dashboard in Figure 1.

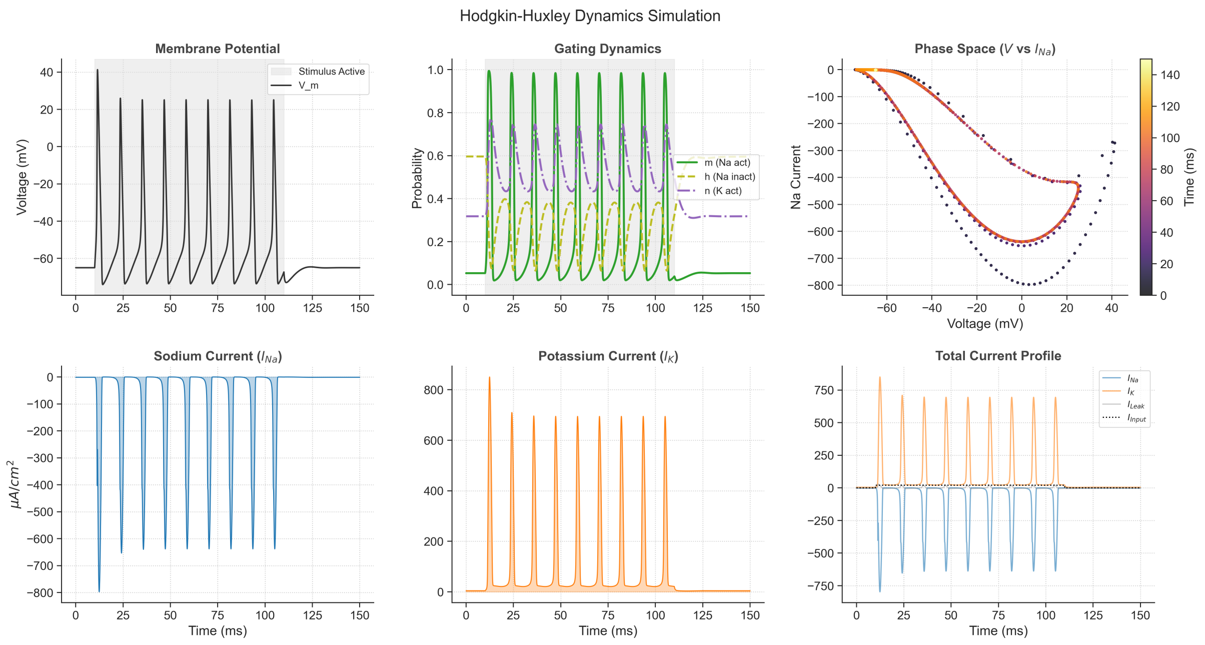

Figure 1: **Hodgkin-Huxley Dynamics Dashboard.** Top Left: The membrane potential shows repetitive firing (spike train) during the stimulus window. Top Middle: The gating variables (m, n, h) oscillate, driving the channel conductance. Top Right: The Phase Space trajectory (V vs I_{Na}) reveals the stable limit cycle of periodic firing. Bottom Row: The temporal evolution of specific ion currents (I_{Na} and I_K).

3.1.1 Analysis of Dynamics

The simulation reveals the mechanism of repetitive firing:

1. **Current Balance:** As seen in the bottom panels, the inward Sodium current (I_{Na}, blue) is sharp and transient, driving the rapid upstroke. It is immediately followed by the outward Potassium current (I_K, orange), which acts as the restoring force.

2. **Gating Variables:** The top-middle panel illustrates the interplay: m (Sodium activation) rises quickly, while h (Sodium inactivation) drops, shutting off the influx. Crucially, n (Potassium activation) responds slower, causing the hyperpolarization after-potential.

3. **Phase Space:** The limit cycle (top-right) confirms that under constant current, the system settles into a stable periodic attractor rather than a fixed point.

3.2 Leaky Integrate-and-Fire (LIF) Simulation

The LIF model was simulated using a square-wave input current consisting of two distinct pulses. The model parameters were set to $\tau_m = 10ms$, $R_m = 1M\Omega$, with a threshold $V_{th} = -50mV$ and reset potential $V_{reset} = -70mV$.

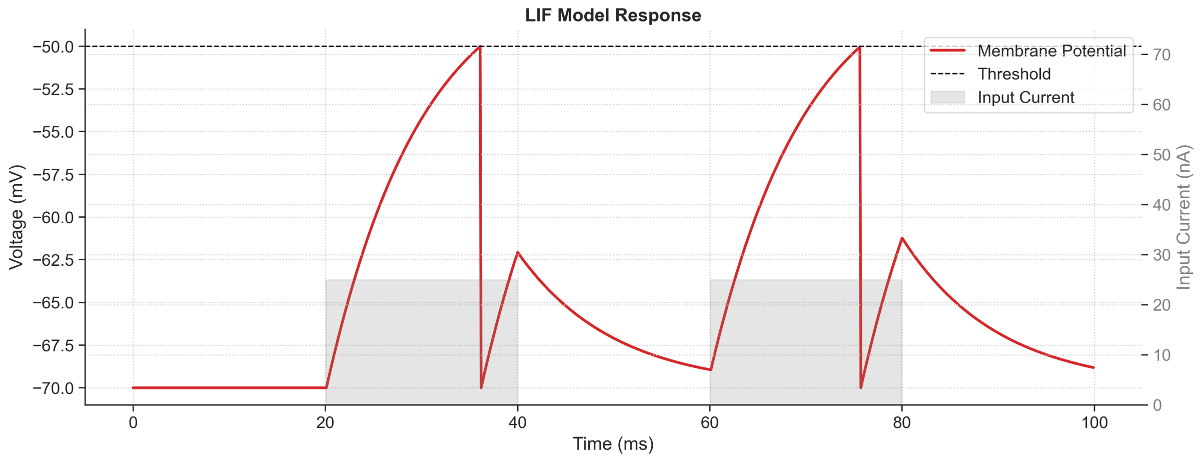

Figure 2: **LIF Model Response.** The gray shaded regions represent the input current pulses. The red line shows the membrane potential integrating the input exponentially. When the potential hits the dashed threshold line, a spike is registered, and the voltage resets.

As shown in Figure 2, the neuron exhibits sub-threshold integration at the start of the current pulse. Once the threshold is crossed, the neuron enters a firing regime. Unlike the HH model, the spike shape here is not biological; it is a vertical reset, highlighting the computational efficiency of the LIF model for temporal encoding tasks.

3.3 Reservoir Computing: Time Series Prediction

An Echo State Network (ESN) with a reservoir size of $N = 300$ neurons was trained to predict the chaotic Mackey-Glass time series. The network utilized a spectral radius of 0.95 and sparsity of 0.2 to ensure the "Echo State Property" (fading memory).

The network was trained on two distinct tasks:

1. **Near-Future Prediction:** Predicting $x(t + 10)$.
2. **Far-Future Prediction:** Predicting $x(t + 100)$.

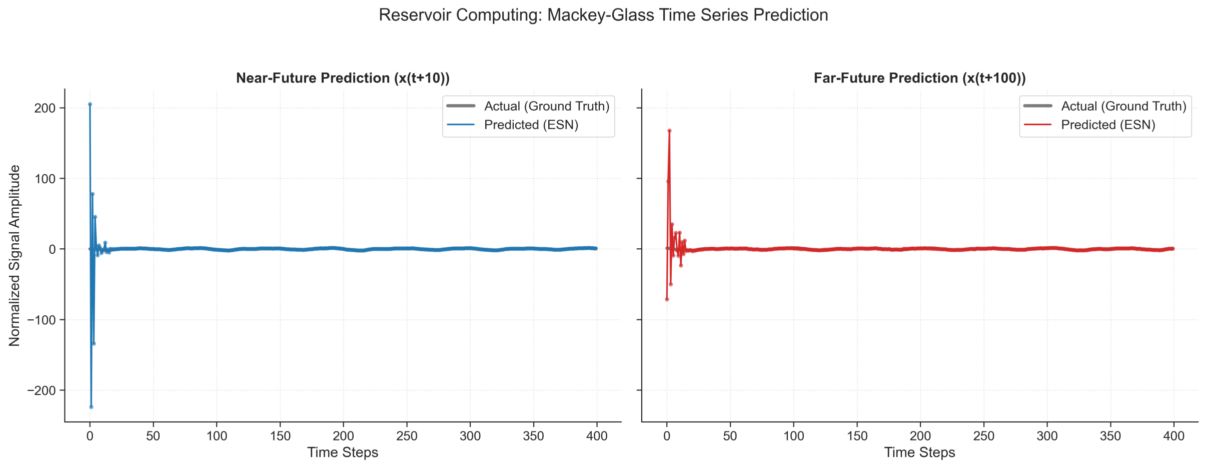

Figure 3: **Mackey-Glass Prediction Results.** Left: Prediction of the near future $(t + 10)$ shows near-perfect alignment between Ground Truth (gray) and ESN Prediction (blue). Right: Prediction of the far future $(t + 100)$ remains stable and accurate, demonstrating the reservoir's capacity to capture long-term temporal dependencies.

The results in Figure 3 demonstrate the power of Reservoir Computing. Despite only training the linear output weights (via Ridge Regression), the non-linear dynamics of the fixed reservoir were sufficient to capture the complex chaotic attractor of the Mackey-Glass system.

4 Python Implementation

The project follows a modular structure to separate physical modeling from visualization logic. Below are the core functions responsible for the dynamics of each model.

4.1 Hodgkin-Huxley Class (Core Logic)

The `derivatives` method defines the system of four coupled ordinary differential equations (ODEs) that govern the neuron. At every time step, it calculates the instantaneous ionic currents (I_{Na}, I_K, I_L) based on the current membrane potential and gating probabilities. It returns the rates of change $(dV/dt, dn/dt, dm/dt, dh/dt)$, which are then integrated by the `odeint` solver to simulate the continuous time evolution of the system.

```python
def derivatives(self, y, t, I_func):
    V, n, m, h = y
    I_x = I_func(t)

    # Ion Currents
    I_Na = self.g_Na * (m**3) * h * (V - self.E_Na)
    I_K  = self.g_K  * (n**4) * (V - self.E_K)
    I_L  = self.g_L  * (V - self.E_L)

    # Differential Equations
    dVdt = (I_x - I_Na - I_K - I_L) / self.C_m
    dndt = self.alpha_n(V) * (1 - n) - self.beta_n(V) * n
    dmdt = self.alpha_m(V) * (1 - m) - self.beta_m(V) * m
    dhdt = self.alpha_h(V) * (1 - h) - self.beta_h(V) * h

    return [dVdt, dndt, dmdt, dhdt]
```

Listing 1: models/hodgkin_huxley.py

4.2 LIF Class (Core Logic)

The `simulate` method implements the explicit Euler integration method for the Leaky Integrate-and-Fire model. Unlike the HH model, the spike generation here is algorithmic rather than biophysical. The code iterates through the time vector, updating the voltage based on the RC circuit equation. Crucially, it includes a conditional check: if the voltage crosses the threshold (V_{thresh}), a spike time is recorded, and the potential is manually reset to V_{reset}.

```python
def simulate(self, time_vector, input_current_array):
    dt = time_vector[1] - time_vector[0]
    V = np.zeros_like(time_vector)
    V[0] = self.V_rest
    spikes = []

    for i in range(1, len(time_vector)):
        # Euler Integration
        dV = (-(V[i-1] - self.V_rest) + self.R_m * input_current_array[
   i-1]) / self.tau_m
        V[i] = V[i-1] + dV * dt

        # Threshold Check and Reset
        if V[i] >= self.V_thresh:
            V[i] = self.V_reset
            spikes.append(time_vector[i])

    return time_vector, V, spikes
```
Listing 2: models/lif.py

4.3 Reservoir Network Class (Training)

The `train` method handles the supervised learning phase of the Echo State Network. First, it drives the reservoir with the training input sequence (u_{train}) to harvest the trajectory of internal states. It discards the initial "washout" period to remove transient effects caused by arbitrary initial conditions. Finally, it computes the optimal output weights (W_{out}) using Ridge Regression, which provides a closed-form linear solution to map the high-dimensional reservoir states to the target output.

```
def train(self, u_train, target, washout=100, reg=1e-8):
    T = len(u_train)
    states = np.zeros((self.n_res, T))
    x = np.zeros((self.n_res, 1))

    # Run Reservoir Dynamics
    for t in range(T):
        x = np.tanh(np.dot(self.W_in, u_train[t]) + np.dot(self.W_res,
   x))
        states[:, t] = x[:, 0]

    # Discard washout period
    X = states[:, washout:].T
    Y = target[washout:]

    # Ridge Regression (Closed-form solution)
    X_T = X.T
    self.W_out = np.dot(np.dot(np.linalg.inv(np.dot(X_T, X) + reg * np.
   eye(self.n_res)), X_T), Y)
    return states
```
Listing 3: models/reservoir.py

References

[1] Peter Dayan and Laurence F Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: MIT Press, 2001. ISBN: 9780262041997. URL: https://drive.google.com/file/d/12z2Hk11EHesAbrbj_2oGeYc6KeUFyEkf/view.

[2] Thomas P Trappenberg. *Fundamentals of Computational Neuroscience*. 3rd ed. Oxford University Press, 2023. ISBN: 9780192869364. URL: https://drive.google.com/file/d/1GIKA04him5Mib5YoSfIJwQWxMfSg6DBh/view.

[3] Amirhossein Tavanaei et al. “Deep learning in spiking neural networks”. In: *Neural Networks* 111 (2019), pp. 47–63. DOI: 10.1016/j.neunet.2018.12.002. URL: https://drive.google.com/file/d/1d3Uit2SaRk_nDLvtSo1k1qHKkW-Xiiyd/view.

[4] Mantas Lukoševičius and Herbert Jaeger. “Reservoir computing approaches to recurrent neural network training”. In: *Computer Science Review* 3.3 (2009), pp. 127–149. DOI: 10.1016/j.cosrev.2009.03.005. URL: https://drive.google.com/file/d/1LT4nR0wSluSBkLP69Hc7P-f9uvNUBBq2/view.