5[1] Doubled the neuron count

Python

```python
import numpy as np
import matplotlib.pyplot as plt

# ========================
# Neuron Model (LIF)
# ========================
def lif_neuron(mem_prev, input_current, dt, tau_mem, threshold,
reset_potential):
    """
    Leaky Integrate-and-Fire neuron model.

    (This basic model does not include a detailed refractory period or complex
bursting dynamics.
    For biological realism, you might add such mechanisms.)
    """
    # Euler integration of the membrane potential
    new_mem = mem_prev + dt/tau_mem * (-mem_prev + input_current)
    # A spike is generated when the membrane potential exceeds the threshold.
    spike = (new_mem >= threshold).astype(float)
    # Reset membrane potential immediately upon spiking
    new_mem = np.where(spike, reset_potential, new_mem)
    return new_mem, spike

# ========================
# STDP Plasticity
# ========================
class STDP:
    def __init__(self, pre_dim, post_dim, tau_plus=20.0, tau_minus=20.0,
                 a_plus=0.1, a_minus=0.1):
        self.tau_plus = tau_plus
        self.tau_minus = tau_minus
        self.a_plus = a_plus
        self.a_minus = a_minus
        self.tr_pre = np.zeros(pre_dim)   # Trace for presynaptic neurons
        self.tr_post = np.zeros(post_dim)   # Trace for postsynaptic neurons

    def update(self, pre_spike, post_spike, dt):
        # Update traces (exponential decay + spike contribution)
        self.tr_pre = self.tr_pre * (1 - dt/self.tau_plus) + pre_spike
        self.tr_post = self.tr_post * (1 - dt/self.tau_minus) + post_spike
        # Compute weight change based on the spike traces
        delta_w = (self.a_plus * np.outer(post_spike, self.tr_pre) -
                   self.a_minus * np.outer(self.tr_post, pre_spike))
```

```
        return delta_w

# ========================
# Simulation Core (with doubled cell numbers)
# ========================
def simulate_model():
    time_steps = 1000
    dt = 0.1  # time step (in seconds)

    # Double the cell numbers compared to the original:
    num_purkinje    = 20   # (was 10)
    num_dcn         = 10   # (was 5)
    num_thalamus    = 10   # (was 5)
    num_motor       = 20   # (was 10)
    num_corticospinal = 40  # (was 20)

    # Neuron parameters
    tau_mem = 7.0
    threshold = 1.0
    reset_potential = 0.0

    # Initialize STDP for the Purkinje -> DCN projection.
    stdp = STDP(pre_dim=num_purkinje, post_dim=num_dcn)

    # Generate random input currents for climbing and mossy fibers.
    climbing_fiber = 7.0 * np.random.rand(num_purkinje, time_steps)
    mossy_fiber    = 7.0 * np.random.rand(num_dcn, time_steps)

    # Allocate arrays for membrane potentials and spikes.
    purk_mem   = np.zeros((num_purkinje, time_steps))
    dcn_mem    = np.zeros((num_dcn, time_steps))
    thal_mem   = np.zeros((num_thalamus, time_steps))
    motor_mem  = np.zeros((num_motor, time_steps))
    cortico_mem = np.zeros((num_corticospinal, time_steps))

    purk_spk   = np.zeros((num_purkinje, time_steps))
    dcn_spk    = np.zeros((num_dcn, time_steps))
    thal_spk   = np.zeros((num_thalamus, time_steps))
    motor_spk  = np.zeros((num_motor, time_steps))
    cortico_spk = np.zeros((num_corticospinal, time_steps))

    # Synaptic weights (note that the dimensions now match the doubled cell
numbers)
    w_purkinje_dcn  = -1.2 * np.ones((num_dcn, num_purkinje))
```

```
    w_dcn_thalamus  = 6.0  * np.ones((num_thalamus, num_dcn))
    w_thalamus_motor = 6.0 * np.ones((num_motor, num_thalamus))
    w_motor_cortico = 6.0 * np.ones((num_corticospinal, num_motor))
    w_motor_dcn     = 0.5 * np.ones((num_dcn, num_motor))

    # This is used to provide feedback to the DCN
    motor_spk_prev = np.zeros(num_motor)

    # Main simulation loop
    for t in range(1, time_steps):
        # Purkinje cells
        purk_mem[:, t], purk_spk[:, t] = lif_neuron(
            purk_mem[:, t-1],
            climbing_fiber[:, t],
            dt, tau_mem, threshold, reset_potential
        )

        # DCN: combine mossy fiber input, inhibitory input from Purkinje, and
feedback from Motor
        feedback = np.dot(w_motor_dcn, motor_spk_prev)
        dcn_input = mossy_fiber[:, t] + np.dot(w_purkinje_dcn, purk_spk[:, t])
+ feedback
        dcn_mem[:, t], dcn_spk[:, t] = lif_neuron(
            dcn_mem[:, t-1],
            dcn_input,
            dt, tau_mem, threshold, reset_potential
        )
        # Update the Purkinje->DCN synaptic weights via STDP
        delta_w = stdp.update(purk_spk[:, t], dcn_spk[:, t], dt)
        w_purkinje_dcn += delta_w

        # Thalamus
        thal_input = np.dot(w_dcn_thalamus, dcn_spk[:, t])
        thal_mem[:, t], thal_spk[:, t] = lif_neuron(
            thal_mem[:, t-1],
            thal_input,
            dt, tau_mem, threshold, reset_potential
        )

        # Motor cortex
        motor_input = np.dot(w_thalamus_motor, thal_spk[:, t])
        motor_mem[:, t], motor_spk[:, t] = lif_neuron(
            motor_mem[:, t-1],
            motor_input,
```

```
            dt, tau_mem, threshold, reset_potential
        )
        motor_spk_prev = motor_spk[:, t].copy()

        # Corticospinal tract
        cortico_input = np.dot(w_motor_cortico, motor_spk[:, t])
        cortico_mem[:, t], cortico_spk[:, t] = lif_neuron(
            cortico_mem[:, t-1],
            cortico_input,
            dt, tau_mem, threshold, reset_potential
        )

    return purk_spk, dcn_spk, thal_spk, motor_spk, cortico_spk, dt

# ========================
# Quantification Functions
# ========================
def quantify_spikes(spike_data, dt):
    """
    Quantify the spike data for a given layer.

    Returns:
        total_spikes: Total number of spikes over all neurons.
        avg_rate: Average firing rate (in Hz) per neuron.
        bursts: Number of bursts detected.

    Burst detection here is very simple: for each neuron, a burst is counted
    when a spike occurs following a period with no spike (i.e. a 0→1
transition).
    """
    total_spikes = np.sum(spike_data)
    num_neurons = spike_data.shape[0]
    sim_time = dt * spike_data.shape[1]  # total simulation time in seconds
    avg_rate = total_spikes / num_neurons / sim_time  # in Hz

    bursts = 0
    # Count burst events: each time a neuron “starts” spiking after silence.
    for neuron in spike_data:
        # Prepend a zero to capture a burst starting at time index 0
        diff = np.diff(np.insert(neuron, 0, 0))
        bursts += np.sum(diff == 1)
    return total_spikes, avg_rate, bursts

# ========================
```

```
# Main Execution and Plotting
# ========================
if __name__ == "__main__":
    # Run the simulation
    p_spk, d_spk, th_spk, m_spk, c_spk, dt = simulate_model()

    # Plot the spike raster for each layer
    plt.figure(figsize=(12, 10))
    layers = [
        ("Purkinje Cells", p_spk),
        ("Deep Cerebellar Nuclei", d_spk),
        ("Thalamus", th_spk),
        ("Motor Cortex", m_spk),
        ("Corticospinal Tract", c_spk)
    ]
    for idx, (name, spk_data) in enumerate(layers):
        plt.subplot(5, 1, idx+1)
        plt.title(name)
        plt.imshow(spk_data, aspect='auto', cmap='binary',
interpolation='none')
        plt.ylabel("Neuron Index")
        if idx == 4:
            plt.xlabel("Time Steps")
    plt.tight_layout()
    plt.savefig("5[1].svg", format="svg")
    plt.show()

    # Compute and print the quantification results for each layer.
    summary_lines = []
    for name, spk_data in layers:
        total_spikes, avg_rate, bursts = quantify_spikes(spk_data, dt)
        line = f"{name} -> Spikes: {int(total_spikes)}, Rate: {avg_rate:.2f}
Hz, Bursts: {int(bursts)}"
        summary_lines.append(line)
        print(line)

    summary_text = "\n".join(summary_lines)

    # Create a figure to display the summary text and save it as an SVG.
    fig, ax = plt.subplots(figsize=(8, 6))
    ax.axis('off')
    ax.text(0.05, 0.5, summary_text, fontsize=12, fontfamily='monospace',
va='center')
    plt.show()
```

```
Unset
Purkinje Cells -> Spikes: 824, Rate: 0.41 Hz, Bursts: 824
Deep Cerebellar Nuclei -> Spikes: 454, Rate: 0.45 Hz, Bursts: 454
Thalamus -> Spikes: 300, Rate: 0.30 Hz, Bursts: 300
Motor Cortex -> Spikes: 300, Rate: 0.15 Hz, Bursts: 300
Corticospinal Tract -> Spikes: 600, Rate: 0.15 Hz, Bursts: 600
```

5[2] Removed noise in climbing fibre and mossy fibre. Also Decreased their weights.

```python
import numpy as np
import matplotlib.pyplot as plt

# ========================
# Neuron Model (LIF)
# ========================
def lif_neuron(mem_prev, input_current, dt, tau_mem, threshold,
reset_potential):
    """
    Leaky Integrate-and-Fire neuron model.

    (This basic model does not include a detailed refractory period or complex
bursting dynamics.
    For biological realism, you might add such mechanisms.)
    """
    # Euler integration of the membrane potential
    new_mem = mem_prev + dt/tau_mem * (-mem_prev + input_current)
    # A spike is generated when the membrane potential exceeds the threshold.
    spike = (new_mem >= threshold).astype(float)
    # Reset membrane potential immediately upon spiking
    new_mem = np.where(spike, reset_potential, new_mem)
    return new_mem, spike

# ========================
# STDP Plasticity
# ========================
class STDP:
    def __init__(self, pre_dim, post_dim, tau_plus=20.0, tau_minus=20.0,
```

```
                 a_plus=0.1, a_minus=0.1):
        self.tau_plus = tau_plus
        self.tau_minus = tau_minus
        self.a_plus = a_plus
        self.a_minus = a_minus
        self.tr_pre = np.zeros(pre_dim)   # Trace for presynaptic neurons
        self.tr_post = np.zeros(post_dim)   # Trace for postsynaptic neurons

    def update(self, pre_spike, post_spike, dt):
        # Update traces (exponential decay + spike contribution)
        self.tr_pre = self.tr_pre * (1 - dt/self.tau_plus) + pre_spike
        self.tr_post = self.tr_post * (1 - dt/self.tau_minus) + post_spike
        # Compute weight change based on the spike traces
        delta_w = (self.a_plus * np.outer(post_spike, self.tr_pre) -
                   self.a_minus * np.outer(self.tr_post, pre_spike))
        return delta_w

# ========================
# Simulation Core (with doubled cell numbers and constant inputs)
# ========================
def simulate_model():
    time_steps = 1000
    dt = 0.1  # time step (in seconds)

    # Double the cell numbers compared to the original:
    num_purkinje     = 20   # (was 10)
    num_dcn          = 10   # (was 5)
    num_thalamus     = 10   # (was 5)
    num_motor        = 20   # (was 10)
    num_corticospinal = 40  # (was 20)

    # Neuron parameters
    tau_mem = 7.0
    threshold = 1.0
    reset_potential = 0.0

    # Initialize STDP for the Purkinje -> DCN projection.
    stdp = STDP(pre_dim=num_purkinje, post_dim=num_dcn)

    # Instead of random noise, use constant input currents for climbing and
mossy fibers.
    # For example, set all values to a constant level.
    constant_climbing_current = 4.0  # you may adjust this constant as needed
    constant_mossy_current    = 4.0  # you may adjust this constant as needed
```

```
    climbing_fiber = constant_climbing_current * np.ones((num_purkinje,
time_steps))
    mossy_fiber    = constant_mossy_current    * np.ones((num_dcn, time_steps))

    # Allocate arrays for membrane potentials and spikes.
    purk_mem    = np.zeros((num_purkinje, time_steps))
    dcn_mem     = np.zeros((num_dcn, time_steps))
    thal_mem    = np.zeros((num_thalamus, time_steps))
    motor_mem   = np.zeros((num_motor, time_steps))
    cortico_mem = np.zeros((num_corticospinal, time_steps))

    purk_spk    = np.zeros((num_purkinje, time_steps))
    dcn_spk     = np.zeros((num_dcn, time_steps))
    thal_spk    = np.zeros((num_thalamus, time_steps))
    motor_spk   = np.zeros((num_motor, time_steps))
    cortico_spk = np.zeros((num_corticospinal, time_steps))

    # Synaptic weights (note that the dimensions now match the doubled cell
numbers)
    w_purkinje_dcn   = -1.2 * np.ones((num_dcn, num_purkinje))
    w_dcn_thalamus   = 6.0  * np.ones((num_thalamus, num_dcn))
    w_thalamus_motor = 6.0  * np.ones((num_motor, num_thalamus))
    w_motor_cortico  = 6.0  * np.ones((num_corticospinal, num_motor))
    w_motor_dcn      = 0.5  * np.ones((num_dcn, num_motor))

    # This is used to provide feedback to the DCN
    motor_spk_prev = np.zeros(num_motor)

    # Main simulation loop
    for t in range(1, time_steps):
        # Purkinje cells
        purk_mem[:, t], purk_spk[:, t] = lif_neuron(
            purk_mem[:, t-1],
            climbing_fiber[:, t],
            dt, tau_mem, threshold, reset_potential
        )

        # DCN: combine mossy fiber input, inhibitory input from Purkinje, and
feedback from Motor
        feedback = np.dot(w_motor_dcn, motor_spk_prev)
        dcn_input = mossy_fiber[:, t] + np.dot(w_purkinje_dcn, purk_spk[:, t])
+ feedback
        dcn_mem[:, t], dcn_spk[:, t] = lif_neuron(
            dcn_mem[:, t-1],
```

```python
            dcn_input,
            dt, tau_mem, threshold, reset_potential
        )
        # Update the Purkinje->DCN synaptic weights via STDP
        delta_w = stdp.update(purk_spk[:, t], dcn_spk[:, t], dt)
        w_purkinje_dcn += delta_w

        # Thalamus
        thal_input = np.dot(w_dcn_thalamus, dcn_spk[:, t])
        thal_mem[:, t], thal_spk[:, t] = lif_neuron(
            thal_mem[:, t-1],
            thal_input,
            dt, tau_mem, threshold, reset_potential
        )

        # Motor cortex
        motor_input = np.dot(w_thalamus_motor, thal_spk[:, t])
        motor_mem[:, t], motor_spk[:, t] = lif_neuron(
            motor_mem[:, t-1],
            motor_input,
            dt, tau_mem, threshold, reset_potential
        )
        motor_spk_prev = motor_spk[:, t].copy()

        # Corticospinal tract
        cortico_input = np.dot(w_motor_cortico, motor_spk[:, t])
        cortico_mem[:, t], cortico_spk[:, t] = lif_neuron(
            cortico_mem[:, t-1],
            cortico_input,
            dt, tau_mem, threshold, reset_potential
        )

    return purk_spk, dcn_spk, thal_spk, motor_spk, cortico_spk, dt

# ========================
# Quantification Functions
# ========================
def quantify_spikes(spike_data, dt):
    """
    Quantify the spike data for a given layer.

    Returns:
        total_spikes: Total number of spikes over all neurons.
        avg_rate: Average firing rate (in Hz) per neuron.
```

```
        bursts: Number of bursts detected.

    Burst detection here is very simple: for each neuron, a burst is counted
    when a spike occurs following a period with no spike (i.e. a 0→1
transition).
    """
    total_spikes = np.sum(spike_data)
    num_neurons = spike_data.shape[0]
    sim_time = dt * spike_data.shape[1]  # total simulation time in seconds
    avg_rate = total_spikes / num_neurons / sim_time  # in Hz

    bursts = 0
    # Count burst events: each time a neuron “starts” spiking after silence.
    for neuron in spike_data:
        # Prepend a zero to capture a burst starting at time index 0
        diff = np.diff(np.insert(neuron, 0, 0))
        bursts += np.sum(diff == 1)
    return total_spikes, avg_rate, bursts

# ========================
# Main Execution and Plotting
# ========================
if __name__ == "__main__":
    # Run the simulation
    p_spk, d_spk, th_spk, m_spk, c_spk, dt = simulate_model()

    # Plot the spike raster for each layer
    plt.figure(figsize=(12, 10))
    layers = [
        ("Purkinje Cells", p_spk),
        ("Deep Cerebellar Nuclei", d_spk),
        ("Thalamus", th_spk),
        ("Motor Cortex", m_spk),
        ("Corticospinal Tract", c_spk)
    ]
    for idx, (name, spk_data) in enumerate(layers):
        plt.subplot(5, 1, idx+1)
        plt.title(name)
        plt.imshow(spk_data, aspect='auto', cmap='binary',
interpolation='none')
        plt.ylabel("Neuron Index")
        if idx == 4:
            plt.xlabel("Time Steps")
    plt.tight_layout()
```

```
    plt.savefig("5[2].svg", format="svg")
    plt.show()

    # Compute and print the quantification results for each layer.
    summary_lines = []
    for name, spk_data in layers:
        total_spikes, avg_rate, bursts = quantify_spikes(spk_data, dt)
        line = f"{name} -> Spikes: {int(total_spikes)}, Rate: {avg_rate:.2f}
Hz, Bursts: {int(bursts)}"
        summary_lines.append(line)
        print(line)

    summary_text = "\n".join(summary_lines)

    # Create a figure to display the summary text and save it as an SVG.
    fig, ax = plt.subplots(figsize=(8, 6))
    ax.axis('off')
    ax.text(0.05, 0.5, summary_text, fontsize=12, fontfamily='monospace',
va='center')
    plt.show()
```

Unset

```
Purkinje Cells -> Spikes: 980, Rate: 0.49 Hz, Bursts: 980
Deep Cerebellar Nuclei -> Spikes: 470, Rate: 0.47 Hz, Bursts: 470
Thalamus -> Spikes: 230, Rate: 0.23 Hz, Bursts: 230
Motor Cortex -> Spikes: 220, Rate: 0.11 Hz, Bursts: 220
Corticospinal Tract -> Spikes: 440, Rate: 0.11 Hz, Bursts: 440
```

5[3] Enhanced model replaces the basic Euler update with a LIF implementation that adds a refractory period, adaptation currents, noise injection, and population-specific parameters.
Also STDP has be replaced with DualSTDP

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

# ========================
# Enhanced Neuron Model (LIF with refractory period and adaptation)
# ========================
def lif_neuron(mem_prev, input_current, dt, params):
    """Leaky Integrate-and-Fire with refractory period, adaptation and noise"""
    if hasattr(params, 'ref_count'):
        if params.ref_count > 0:
            params.ref_count -= 1
            return mem_prev, 0, params.adapt_current

    noise = params.noise_scale * np.random.randn(*mem_prev.shape)
    new_mem = mem_prev + dt/params.tau_mem * (-mem_prev + input_current -
params.adapt_current) + noise
    spike = (new_mem >= params.threshold).astype(float)

    if spike:
        new_mem = params.reset_potential
        params.ref_count = int(params.ref_period/dt)
        params.adapt_current = params.adapt_current + params.adapt_increment
    else:
        params.adapt_current *= params.adapt_decay

    return new_mem, spike, params.adapt_current

class NeuronParams:
    def __init__(self, population):
        self.population = population
        # Population-specific parameters
        params = {
            'Purkinje': {'tau_mem': 10.0, 'threshold': -55.0, 'reset': -70.0,
'noise': 0.3},
            'DCN': {'tau_mem': 20.0, 'threshold': -50.0, 'reset': -65.0,
'noise': 0.2},
```

```
            'Thalamus': {'tau_mem': 25.0, 'threshold': -54.0, 'reset': -65.0,
'noise': 0.2},
            'Motor': {'tau_mem': 20.0, 'threshold': -52.0, 'reset': -65.0,
'noise': 0.25},
            'Corticospinal': {'tau_mem': 30.0, 'threshold': -53.0, 'reset':
-65.0, 'noise': 0.2}
        }
        p = params[population]

        self.tau_mem = p['tau_mem']
        self.threshold = p['threshold']
        self.reset_potential = p['reset']
        self.noise_scale = p['noise']
        self.ref_period = 2.0  # 2ms refractory period
        self.ref_count = 0
        self.adapt_current = 0
        self.adapt_increment = 0.1
        self.adapt_decay = 0.95

# ========================
# Enhanced STDP Plasticity
# ========================
class DualSTDP:
    def __init__(self, pre_dim, post_dim):
        self.tau_plus = 20.0
        self.tau_minus = 30.0  # Asymmetric timing windows
        self.a_plus = 0.002
        self.a_minus = 0.001
        self.tr_pre = np.zeros(pre_dim)
        self.tr_post = np.zeros(post_dim)

    def update(self, pre_spike, post_spike, error_signal, dt):
        self.tr_pre = self.tr_pre * np.exp(-dt/self.tau_plus) + pre_spike
        self.tr_post = self.tr_post * np.exp(-dt/self.tau_minus) + post_spike
        delta_w = (self.a_plus * np.outer(post_spike, self.tr_pre) -
                   self.a_minus * np.outer(self.tr_post, pre_spike))
        return delta_w * (1 + 0.1 * error_signal)  # Reduced error sensitivity

# ========================
# Simulation Parameters
# ========================
time_steps = 1000
dt = 0.1
num_purkinje = 20
```

```python
num_dcn = 10
num_thalamus = 10
num_motor = 20
num_corticospinal = 40

neuron_structure = [
    ('Purkinje', num_purkinje),
    ('DCN', num_dcn),
    ('Thalamus', num_thalamus),
    ('Motor', num_motor),
    ('Corticospinal', num_corticospinal)
]

# ========================
# Realistic Input Patterns
# ========================
def generate_motor_command(t):
    base = 0.02 * np.sin(2*np.pi*t/1000)  # Reduced amplitude
    noise = 0.01 * np.random.randn(num_motor)
    return base + noise

def generate_poisson_spikes(rate, size, steps):
    return (np.random.rand(size, steps) < rate * dt * 0.001).astype(float)

# Reduced firing rates
climbing_fiber = generate_poisson_spikes(1, num_purkinje, time_steps)  # 1 Hz
mossy_fiber = generate_poisson_spikes(10, num_dcn, time_steps)  # 10 Hz

# ========================
# Simulation Core
# ========================
def simulate_model():
    # Initialize neuron parameters
    neuron_params = {name: [NeuronParams(name) for _ in range(n_neurons)]
                    for name, n_neurons in neuron_structure}

    membranes = {name: np.full((n_neurons, time_steps),
params[0].reset_potential)
                for (name, n_neurons), params in zip(neuron_structure,
neuron_params.values())}

    spikes = {name: np.zeros((n_neurons, time_steps))
             for name, n_neurons in neuron_structure}
```

```
    # Initialize weights with Dale's principle (Purkinje inhibitory, others
excitatory)
    weights = {
        'Purkinje-DCN': -0.5 * np.ones((num_dcn, num_purkinje)) +
                        0.1 * np.random.randn(num_dcn, num_purkinje),
        'DCN-Thalamus': 0.2 * np.abs(np.random.randn(num_thalamus, num_dcn)),
        'Thalamus-Motor': 0.15 * np.abs(np.random.randn(num_motor,
num_thalamus)),
        'Motor-Corticospinal': 0.1 * np.abs(np.random.randn(num_corticospinal,
num_motor))
    }

    stdp = DualSTDP(num_purkinje, num_dcn)
    background_noise = 0.05  # Reduced background noise

    for t in range(1, time_steps):
        # Update each population
        for name, params in neuron_params.items():
            for i, p in enumerate(params):
                if name == 'Purkinje':
                    current = (climbing_fiber[i,t] +
                             0.05*generate_motor_command(t)[0] +
                             background_noise * np.random.randn())
                elif name == 'DCN':
                    current = (mossy_fiber[i,t] +
                             np.dot(weights['Purkinje-DCN'][i],
spikes['Purkinje'][:,t-1]) +
                             background_noise * np.random.randn())
                elif name == 'Thalamus':
                    current = (np.dot(weights['DCN-Thalamus'][i],
spikes['DCN'][:,t-1]) +
                             background_noise * np.random.randn())
                elif name == 'Motor':
                    current = (np.dot(weights['Thalamus-Motor'][i],
spikes['Thalamus'][:,t-1]) +
                             generate_motor_command(t)[i] +
                             background_noise * np.random.randn())
                else:  # Corticospinal
                    current = (np.dot(weights['Motor-Corticospinal'][i],
spikes['Motor'][:,t-1]) +
                             background_noise * np.random.randn())

                membranes[name][i,t], spikes[name][i,t], p.adapt_current =
lif_neuron(
```

```
                    membranes[name][i,t-1], current, dt, p)

        # Update weights with STDP
        if t % 10 == 0:  # Reduce computation frequency
            error = np.mean(spikes['Corticospinal'][:,t-1]) - 0.01
            delta_w = stdp.update(spikes['Purkinje'][:,t], spikes['DCN'][:,t],
error, dt)
            weights['Purkinje-DCN'] = np.clip(weights['Purkinje-DCN'] +
delta_w, -1.0, -0.1)

    return spikes, weights

# Run simulation and analysis as before


# ========================
# Analysis & Visualization
# ========================
def analyze_output(spikes):
    metrics = {}
    for name, spk_data in spikes.items():
        total_spikes = np.sum(spk_data)
        firing_rate = total_spikes / (time_steps*dt/1000) / spk_data.shape[0]

        bursts = 0
        for neuron in spk_data:
            peaks, _ = find_peaks(neuron, height=0.5, distance=5)
            bursts += len(peaks)

        metrics[name] = {
            'Spikes': int(total_spikes),
            'Rate': f"{firing_rate:.2f} Hz",
            'Bursts': bursts
        }
    return metrics

# Run simulation and visualize
spikes, weights = simulate_model()
metrics = analyze_output(spikes)

print("\n=== Network Output Metrics ===")
for name, vals in metrics.items():
    print(f"{name} -> Spikes: {vals['Spikes']}, Rate: {vals['Rate']}, Bursts:
{vals['Bursts']}")
```

```python
plt.figure(figsize=(15, 10))
for idx, (name, spk_data) in enumerate(spikes.items()):
    plt.subplot(len(spikes), 1, idx + 1)
    plt.eventplot([np.where(row)[0] for row in spk_data],
                 color='black', linelengths=0.8)
    plt.ylabel(name)
    if idx == len(spikes) - 1:
        plt.xlabel('Time (ms)')
plt.suptitle('Neural Network Activity')
plt.tight_layout()
plt.savefig("5[3].svg", format="svg")
plt.show()
```

```
Unset
Purkinje -> Spikes: 445, Rate: 222.50 Hz, Bursts: 445
DCN -> Spikes: 134, Rate: 134.00 Hz, Bursts: 134
Thalamus -> Spikes: 146, Rate: 146.00 Hz, Bursts: 146
Motor -> Spikes: 303, Rate: 151.50 Hz, Bursts: 303
Corticospinal -> Spikes: 481, Rate: 120.25 Hz, Bursts: 480
```

3[1] Doubled neuron count and removed noise in mossy and climbing fibres

Python
```python
import numpy as np
import matplotlib.pyplot as plt

# ------------------------------
# Simulation and Model Parameters
# ------------------------------
time_steps = 1000  # Total simulation time steps
dt = 0.1           # Time step size in ms

# Doubling the number of cells:
num_purkinje      = 20   # (was 10)
num_dcn           = 10   # (was 5)
num_thalamus      = 10   # (was 5)
num_motor_cortex  = 20   # (was 10)
num_corticospinal = 40   # (was 20)

# Neuron model parameters
tau_mem = 10.0          # Membrane time constant (ms)
threshold = 1.0         # Spiking threshold
resting_potential = 0.0 # Resting membrane potential
reset_potential = 0.0   # Reset potential after spike

# Best configuration parameters (from optimization results)
w_purkinje_dcn_val        = 2.3   # Inhibitory weight from Purkinje to DCN
w_dcn_thalamus_val        = 2.1   # Excitatory weight from DCN to Thalamus
w_thalamus_motor_val      = 3.0   # Excitatory weight from Thalamus to Motor
Cortex
w_motor_corticospinal_val = 2.6   # Excitatory weight from Motor Cortex to
Corticospinal tract

# Synaptic weights
w_purkinje_dcn        = -w_purkinje_dcn_val * np.ones((num_dcn, num_purkinje))
# Increase excitatory drive downstream by scaling up the weights:
w_dcn_thalamus        = (w_dcn_thalamus_val * 1.2) * np.ones((num_thalamus,
num_dcn))
w_thalamus_motor      = (w_thalamus_motor_val * 1.5) *
np.ones((num_motor_cortex, num_thalamus))
```

```
# Increase the effective weight for corticospinal neurons (from 1.5 to 2.0
multiplier):
w_motor_corticospinal = (w_motor_corticospinal_val * 2.0) *
np.ones((num_corticospinal, num_motor_cortex))

# ------------------------------
# Input Setup (No Artificial Noise)
# ------------------------------
# Adjust constant inputs to help balance network activity.
constant_input_purkinje = 8.0    # Lowered drive for Purkinje cells
constant_input_dcn      = 10.0   # Constant drive for DCN
climbing_fiber_input = constant_input_purkinje * np.ones((num_purkinje,
time_steps))
mossy_fiber_input    = constant_input_dcn      * np.ones((num_dcn, time_steps))

# Baseline drives for Motor Cortex and Corticospinal tract:
baseline_motor_drive = 0.5       # Baseline drive to Motor Cortex (as before)
baseline_corticospinal = 1.0     # NEW: Baseline drive to Corticospinal neurons

# ------------------------------
# State Variables: Membrane Potentials and Spikes
# ------------------------------
purkinje_membrane      = np.zeros((num_purkinje, time_steps))
dcn_membrane           = np.zeros((num_dcn, time_steps))
thalamus_membrane      = np.zeros((num_thalamus, time_steps))
motor_membrane         = np.zeros((num_motor_cortex, time_steps))
corticospinal_membrane = np.zeros((num_corticospinal, time_steps))

purkinje_spikes      = np.zeros((num_purkinje, time_steps))
dcn_spikes           = np.zeros((num_dcn, time_steps))
thalamus_spikes      = np.zeros((num_thalamus, time_steps))
motor_spikes         = np.zeros((num_motor_cortex, time_steps))
corticospinal_spikes = np.zeros((num_corticospinal, time_steps))

# ------------------------------
# Simulation Loop
# ------------------------------
for t in range(1, time_steps):
    # --- Purkinje Cells ---
    purkinje_input = climbing_fiber_input[:, t]
    purkinje_membrane[:, t] = purkinje_membrane[:, t - 1] + dt / tau_mem * (
        -purkinje_membrane[:, t - 1] + purkinje_input)
    purkinje_spikes[:, t] = (purkinje_membrane[:, t] >=
threshold).astype(float)
```

```
    purkinje_membrane[purkinje_spikes[:, t] == 1, t] = reset_potential

    # --- Deep Cerebellar Nuclei (DCN) ---
    dcn_input = mossy_fiber_input[:, t] + np.dot(w_purkinje_dcn,
purkinje_spikes[:, t])
    dcn_membrane[:, t] = dcn_membrane[:, t - 1] + dt / tau_mem * (
        -dcn_membrane[:, t - 1] + dcn_input)
    dcn_spikes[:, t] = (dcn_membrane[:, t] >= threshold).astype(float)
    dcn_membrane[dcn_spikes[:, t] == 1, t] = reset_potential

    # --- Thalamus ---
    thalamus_input = np.dot(w_dcn_thalamus, dcn_spikes[:, t])
    thalamus_membrane[:, t] = thalamus_membrane[:, t - 1] + dt / tau_mem * (
        -thalamus_membrane[:, t - 1] + thalamus_input)
    thalamus_spikes[:, t] = (thalamus_membrane[:, t] >=
threshold).astype(float)
    thalamus_membrane[thalamus_spikes[:, t] == 1, t] = reset_potential

    # --- Motor Cortex ---
    motor_input = np.dot(w_thalamus_motor, thalamus_spikes[:, t]) +
baseline_motor_drive
    motor_membrane[:, t] = motor_membrane[:, t - 1] + dt / tau_mem * (
        -motor_membrane[:, t - 1] + motor_input)
    motor_spikes[:, t] = (motor_membrane[:, t] >= threshold).astype(float)
    motor_membrane[motor_spikes[:, t] == 1, t] = reset_potential

    # --- Corticospinal Tract ---
    # Now add a baseline drive to help push these neurons over threshold
    corticospinal_input = np.dot(w_motor_corticospinal, motor_spikes[:, t]) +
baseline_corticospinal
    corticospinal_membrane[:, t] = corticospinal_membrane[:, t - 1] + dt /
tau_mem * (
        -corticospinal_membrane[:, t - 1] + corticospinal_input)
    corticospinal_spikes[:, t] = (corticospinal_membrane[:, t] >=
threshold).astype(float)
    corticospinal_membrane[corticospinal_spikes[:, t] == 1, t] =
reset_potential

# ------------------------------
# Spike Quantification
# ------------------------------
def quantify_spikes(spike_data, dt, region_name, num_neurons):
    """
    Quantifies spikes for a given region.
```

```
    Parameters:
      - spike_data: 2D array of spikes (neurons x time)
      - dt: time step size in ms
      - region_name: name of the region (string)
      - num_neurons: number of neurons in the region

    Returns:
      (total_spikes, average_rate (Hz), bursts)

    Note: Burst detection here is rudimentary; each 0→1 transition per neuron
is counted.
    """
    sim_time_sec = time_steps * dt / 1000.0  # convert ms to seconds
    total_spikes = np.sum(spike_data)
    avg_rate = total_spikes / num_neurons / sim_time_sec  # in Hz
    bursts = 0
    for neuron in spike_data:
        transitions = np.diff(np.insert(neuron, 0, 0))
        bursts += np.sum(transitions == 1)
    return total_spikes, avg_rate, bursts

# Quantify for each region
purkinje_stats      = quantify_spikes(purkinje_spikes, dt, "Purkinje Cells",
num_purkinje)
dcn_stats           = quantify_spikes(dcn_spikes, dt, "DCN", num_dcn)
thalamus_stats      = quantify_spikes(thalamus_spikes, dt, "Thalamus",
num_thalamus)
motor_cortex_stats  = quantify_spikes(motor_spikes, dt, "Motor Cortex",
num_motor_cortex)
corticospinal_stats = quantify_spikes(corticospinal_spikes, dt, "Corticospinal
Tract", num_corticospinal)

# Create summary text
summary_lines = [
    f"Purkinje Cells -> Spikes: {int(purkinje_stats[0])}, Rate:
{purkinje_stats[1]:.2f} Hz, Bursts: {int(purkinje_stats[2])}",
    f"DCN -> Spikes: {int(dcn_stats[0])}, Rate: {dcn_stats[1]:.2f} Hz, Bursts:
{int(dcn_stats[2])}",
    f"Thalamus -> Spikes: {int(thalamus_stats[0])}, Rate:
{thalamus_stats[1]:.2f} Hz, Bursts: {int(thalamus_stats[2])}",
    f"Motor Cortex -> Spikes: {int(motor_cortex_stats[0])}, Rate:
{motor_cortex_stats[1]:.2f} Hz, Bursts: {int(motor_cortex_stats[2])}",
```

```
    f"Corticospinal Tract -> Spikes: {int(corticospinal_stats[0])}, Rate:
{corticospinal_stats[1]:.2f} Hz, Bursts: {int(corticospinal_stats[2])}"
]
summary_text = "\n".join(summary_lines)
print(summary_text)

# ------------------------------
# Save Summary as SVG
# ------------------------------
fig, ax = plt.subplots(figsize=(8, 6))
ax.axis('off')
ax.text(0.05, 0.5, summary_text, fontsize=12, fontfamily='monospace',
va='center')
plt.savefig("3[1].svg", format="svg")
plt.close(fig)

# ------------------------------
# Plot Raster Plots for Each Region
# ------------------------------
plt.figure(figsize=(12, 10))

plt.subplot(5, 1, 1)
plt.title("Purkinje Cells")
plt.imshow(purkinje_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 2)
plt.title("Deep Cerebellar Nuclei (DCN)")
plt.imshow(dcn_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 3)
plt.title("Thalamus")
plt.imshow(thalamus_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 4)
plt.title("Motor Cortex")
plt.imshow(motor_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 5)
plt.title("Corticospinal Tract")
plt.imshow(corticospinal_spikes, aspect='auto', cmap='binary')
```

```
plt.ylabel("Neuron Index")
plt.xlabel("Time Steps")

plt.tight_layout()
plt.show()
```

```
Unset
Purkinje Cells -> Spikes: 1420, Rate: 710.00 Hz, Bursts: 1420
DCN -> Spikes: 590, Rate: 590.00 Hz, Bursts: 590
Thalamus -> Spikes: 90, Rate: 90.00 Hz, Bursts: 90
Motor Cortex -> Spikes: 80, Rate: 40.00 Hz, Bursts: 80
Corticospinal Tract -> Spikes: 160, Rate: 40.00 Hz, Bursts: 160
```

3[2] Double Neurons

```python
import numpy as np
import matplotlib.pyplot as plt

# Simulation parameters
time_steps = 1000  # Total simulation time steps
dt = 0.1  # Time step size (ms)
num_purkinje = 20  # Doubled number of Purkinje cells
num_dcn = 10  # Doubled number of DCN neurons
num_thalamus = 10  # Doubled number of thalamic neurons
num_motor_cortex = 20  # Doubled number of motor cortex neurons
num_corticospinal = 40  # Doubled number of corticospinal tract neurons

# Neuron model parameters
tau_mem = 10.0  # Membrane time constant (ms)
threshold = 1.0  # Spiking threshold
resting_potential = 0.0  # Resting membrane potential
reset_potential = 0.0  # Reset potential after spike

# Best configuration parameters (from optimization results)
w_purkinje_dcn_val = 2.3
w_dcn_thalamus_val = 2.1
w_thalamus_motor_val = 3.0
w_motor_corticospinal_val = 2.6
noise_amplitude_val = 2.0

# Synaptic weights
w_purkinje_dcn = -w_purkinje_dcn_val * np.ones((num_dcn, num_purkinje))
w_dcn_thalamus = w_dcn_thalamus_val * np.ones((num_thalamus, num_dcn))
w_thalamus_motor = w_thalamus_motor_val * np.ones((num_motor_cortex,
num_thalamus))
w_motor_corticospinal = w_motor_corticospinal_val * np.ones((num_corticospinal,
num_motor_cortex))

# Inputs: scaled to increase excitability
climbing_fiber_input = 10.0 * np.random.rand(num_purkinje, time_steps)
mossy_fiber_input = 10.0 * np.random.rand(num_dcn, time_steps)

# Neuron states: membrane potentials and spikes
```

```
purkinje_membrane = np.zeros((num_purkinje, time_steps))
dcn_membrane = np.zeros((num_dcn, time_steps))
thalamus_membrane = np.zeros((num_thalamus, time_steps))
motor_membrane = np.zeros((num_motor_cortex, time_steps))
corticospinal_membrane = np.zeros((num_corticospinal, time_steps))

purkinje_spikes = np.zeros((num_purkinje, time_steps))
dcn_spikes = np.zeros((num_dcn, time_steps))
thalamus_spikes = np.zeros((num_thalamus, time_steps))
motor_spikes = np.zeros((num_motor_cortex, time_steps))
corticospinal_spikes = np.zeros((num_corticospinal, time_steps))

# Simulation loop
for t in range(1, time_steps):
    # Purkinje cells
    purkinje_input = climbing_fiber_input[:, t]
    purkinje_membrane[:, t] = purkinje_membrane[:, t - 1] + dt / tau_mem *
(-purkinje_membrane[:, t - 1] + purkinje_input)
    purkinje_spikes[:, t] = (purkinje_membrane[:, t] >=
threshold).astype(float)
    purkinje_membrane[purkinje_spikes[:, t] == 1, t] = reset_potential

    # Deep cerebellar nuclei (DCN)
    dcn_input = mossy_fiber_input[:, t] + np.dot(w_purkinje_dcn,
purkinje_spikes[:, t])
    dcn_membrane[:, t] = dcn_membrane[:, t - 1] + dt / tau_mem *
(-dcn_membrane[:, t - 1] + dcn_input)
    dcn_spikes[:, t] = (dcn_membrane[:, t] >= threshold).astype(float)
    dcn_membrane[dcn_spikes[:, t] == 1, t] = reset_potential

    # Thalamus
    thalamus_input = np.dot(w_dcn_thalamus, dcn_spikes[:, t]) +
noise_amplitude_val * np.random.rand(num_thalamus)
    thalamus_membrane[:, t] = thalamus_membrane[:, t - 1] + dt / tau_mem *
(-thalamus_membrane[:, t - 1] + thalamus_input)
    thalamus_spikes[:, t] = (thalamus_membrane[:, t] >=
threshold).astype(float)
    thalamus_membrane[thalamus_spikes[:, t] == 1, t] = reset_potential

    # Motor cortex
    motor_input = np.dot(w_thalamus_motor, thalamus_spikes[:, t]) +
noise_amplitude_val * np.random.rand(num_motor_cortex)
    motor_membrane[:, t] = motor_membrane[:, t - 1] + dt / tau_mem *
(-motor_membrane[:, t - 1] + motor_input)
```

```
    motor_spikes[:, t] = (motor_membrane[:, t] >= threshold).astype(float)
    motor_membrane[motor_spikes[:, t] == 1, t] = reset_potential

    # Corticospinal tract
    corticospinal_input = np.dot(w_motor_corticospinal, motor_spikes[:, t]) +
noise_amplitude_val * np.random.rand(num_corticospinal)
    corticospinal_membrane[:, t] = corticospinal_membrane[:, t - 1] + dt /
tau_mem * (-corticospinal_membrane[:, t - 1] + corticospinal_input)
    corticospinal_spikes[:, t] = (corticospinal_membrane[:, t] >=
threshold).astype(float)
    corticospinal_membrane[corticospinal_spikes[:, t] == 1, t] =
reset_potential

# Quantization logic for spike statistics
def calculate_stats(spike_array):
    total_spikes_per_neuron = spike_array.sum(axis=1)
    total_spikes_all_neurons = total_spikes_per_neuron.sum()
    firing_rate_hz_per_neuron = total_spikes_per_neuron / (time_steps * dt) *
1000
    avg_firing_rate_hz_all_neurons = firing_rate_hz_per_neuron.mean()

    return total_spikes_all_neurons, avg_firing_rate_hz_all_neurons

purkinje_stats = calculate_stats(purkinje_spikes)
dcn_stats = calculate_stats(dcn_spikes)
thalamus_stats = calculate_stats(thalamus_spikes)
motor_stats = calculate_stats(motor_spikes)
corticospinal_stats = calculate_stats(corticospinal_spikes)

print(f"Purkinje Cells -> Spikes: {purkinje_stats[0]}, Rate:
{purkinje_stats[1]:.2f} Hz")
print(f"DCN Neurons -> Spikes: {dcn_stats[0]}, Rate: {dcn_stats[1]:.2f} Hz")
print(f"Thalamus -> Spikes: {thalamus_stats[0]}, Rate: {thalamus_stats[1]:.2f}
Hz")
print(f"Motor Cortex -> Spikes: {motor_stats[0]}, Rate: {motor_stats[1]:.2f}
Hz")
print(f"Corticospinal Tract -> Spikes: {corticospinal_stats[0]}, Rate:
{corticospinal_stats[1]:.2f} Hz")

# Plotting results as raster plots and saving as SVG file
plt.figure(figsize=(12, 8))
plt.subplot(5, 1, 1)
plt.title("Purkinje Cells")
plt.imshow(purkinje_spikes, aspect='auto', cmap='binary')
```

```
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 2)
plt.title("Deep Cerebellar Nuclei (DCN)")
plt.imshow(dcn_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 3)
plt.title("Thalamus")
plt.imshow(thalamus_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 4)
plt.title("Motor Cortex")
plt.imshow(motor_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")

plt.subplot(5, 1, 5)
plt.title("Corticospinal Tract")
plt.imshow(corticospinal_spikes, aspect='auto', cmap='binary')
plt.ylabel("Neuron Index")
plt.xlabel("Time Steps")

plt.tight_layout()
plt.savefig("3[2].svg", format="svg")
plt.show()
```

Unset

```
Purkinje Cells -> Spikes: 866.0, Rate: 433.00 Hz
DCN Neurons -> Spikes: 233.0, Rate: 233.00 Hz
Thalamus -> Spikes: 86.0, Rate: 86.00 Hz
Motor Cortex -> Spikes: 118.0, Rate: 59.00 Hz
Corticospinal Tract -> Spikes: 271.0, Rate: 67.75 Hz
```