Modular Network Simulation with Izhikevich Neurons

📖 Chapters

- Class Architecture
  - Attributes
  - Methods
- Usage and Execution
  - Simulation Flow
  - Instructions to Run
- Visualizations
  - Connection Matrices
  - Raster Plots
  - Mean Firing Rates
  - Network Visualization

Class Architecture

We decided to create a class (`ModularNetwork`) that represents a modular network of Izhikevich neurons, as there are many parameters that are shared between functions. We divided the code into several private methods and helper functions to improve readability and reusability. The public methods are `generate_modular_network`, to create the basic network structure; `run_simulation`, to simulate the behaviour of the network over time, and all the different necessary plotting functions. An overview of the class attributes and methods goes as follows:

Attributes

- `p`: rewiring probability for the small-world network.
- `params`: dictionary containing all the parameters for the simulation. They are unpacked to class attributes. These parameters are:
  - `NUMBER_OF_MODULES`: number of modules in the network.
  - `EXCITATORY_PER_MODULE`: number of excitatory neurons per module.
  - `INHIBITORY_NEURONS`: total number of inhibitory neurons.
  - `CONNECTIONS_PER_MODULE`: number of connections per module.
  - `excitatory_iz_neuron`: parameters for excitatory Izhikevich neurons, containing `a`, `b`, `c`, `d` and their respective noise values `a_r`, `b_r`, `c_r`, `d_r`.
  - `inhibitory_iz_neuron`: parameters for inhibitory Izhikevich neurons, containing `a`, `b`, `c`, `d` and their respective noise values `a_r`, `b_r`, `c_r`, `d_r`.
- `network`: adjacency matrix representing the connections between neurons. It is created in the `generate_modular_network` method.
- `W`: connectivity matrix with synaptic weights between neurons. It is created in the `generate_modular_network` method.
- `D`: delay matrix with synaptic delays between neurons. It is created in the `generate_modular_network` method.

Methods

- `__init__(self, p, params)`: constructor method that initializes the class attributes. It takes the rewiring probability `p` and a dictionary of parameters `params` as input.
- `_generate_neuron_parameters(self) -> tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]`: private method that generates the parameters for each neuron in the network, adding some noise to the base parameters. It returns four arrays: `a`, `b`, `c`, and `d`.
- `_generate_ee_connections(self) -> list[tuple[int, int]]`: private method that creates excitatory-excitatory connections using the Watts-Strogatz small-world model. It returns a list of tuples representing the connections.
- `_generate_ei_connections(self)`: private method that creates excitatory-inhibitory connections.
- `_generate_ie_connections(self)`: private method that creates inhibitory-excitatory connections.
- `_generate_ii_connections(self)`: private method that creates inhibitory-inhibitory connections.
- `_rewire_ee_connections(self) -> int`: private method that rewires the excitatory-excitatory connections based on the rewiring probability `p`. It returns the number of connections that were rewired.
- `generate_modular_network(self) -> IzNetwork`: public method that generates the modular network by calling the private methods to create connections and rewiring, using the provided IzNetwork class. It returns the instance of IzNetwork that is created.
- `run_simulation(self, sim_time) -> list[tuple[int, int]]`: public method that simulates the network dynamics over a specified simulation time `sim_time`. It returns a list of tuples representing the spike times and neuron indices.
- `connectivity_matrix`(self, excitatory_only, title, save_plot, plot_filename): public method that plots the connectivity matrix of the network. If `excitatory_only` is True, it only shows excitatory neurons.
- `raster_plot(self, spikes, sim_time, excitatory_only, y0_on_top, save_plot, plot_filename)`: public method that creates a raster plot of the spike times. If `excitatory_only` is True, it only shows excitatory neurons. If `y0_on_top` is True, it inverts the y-axis.
- `mean_firing_rate(self, activations, sim_time, window_size, step_size, include_inhibitory, save_plot, plot_filename)`: public method that calculates and plots the mean firing rate over time. If `include_inhibitory` is True, it includes inhibitory neurons in the calculation.
- `visualize_simulation(self, activations, sim_time, save_animation, animation_filename, fps)`: public method that shows the spatial positions of neurons and their activations over time as an animation.

Usage and Execution

Simulation Flow

The overall program flow begins in the `main()` method, which initializes the parameters and creates a `ModularNetwork` instance for each rewiring probability `p`.
For each instance:

- `generate_modular_network()` builds the network by calling the internal helper methods for connection generation and neuron parameter setup.

- `run_simulation()` then uses the generated `IzNetwork` to simulate neuronal activity over time and record spike events.
- Finally, the plotting methods (`connectivity_matrix`, `raster_plot`, `mean_firing_rate`, and optionally `visualize_simulation`) visualize the network's structure and dynamics.

Instructions to Run

1. Ensure that the file `iznetwork.py` is available and located in the same directory as `modular_network.py`.

2. Run the simulation. You have two main options:

   - **Default Mode (Save Static Plots):** Run the script without any flags. The simulation will run to completion and automatically create a folder named `plots/` containing all the static visualizations (connection matrices, raster plots, and firing rate plots).

     ```
     python3 modular_network.py
     ```

   - **Visualization Mode (Live Animation):** To watch a live animation of the neurons firing during the simulation, use the `-v` or `--visualize` flag.

     ```
     python3 modular_network.py --visualize
     ```

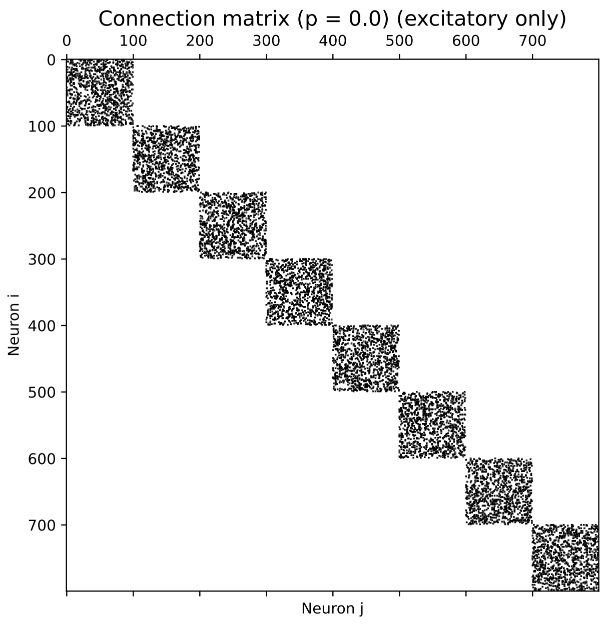
     Note: When this flag is set, you will see the animation at runtime, but the program will **not** save the animations to the `plots/` folder. The live animation itself is not saved as a GIF because the process is very time-consuming.

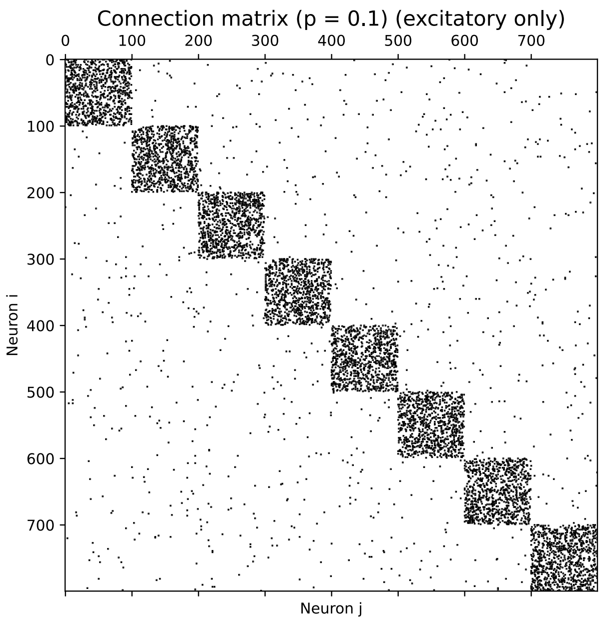
     You can see pre-rendered examples of these animations by scrolling to the bottom of the `README.md` on our GitHub repository.

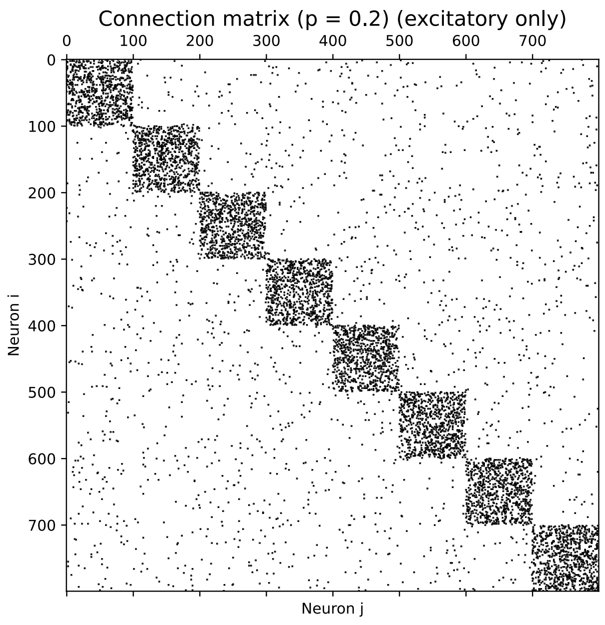
3. The program will automatically create a folder named plots/, which will contain all generated visualizations (connection matrices, raster plots, and firing rate plots).

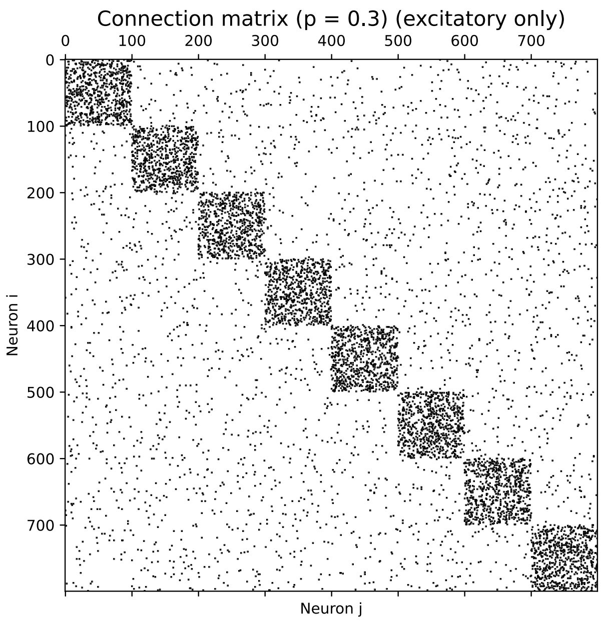
Visualizations

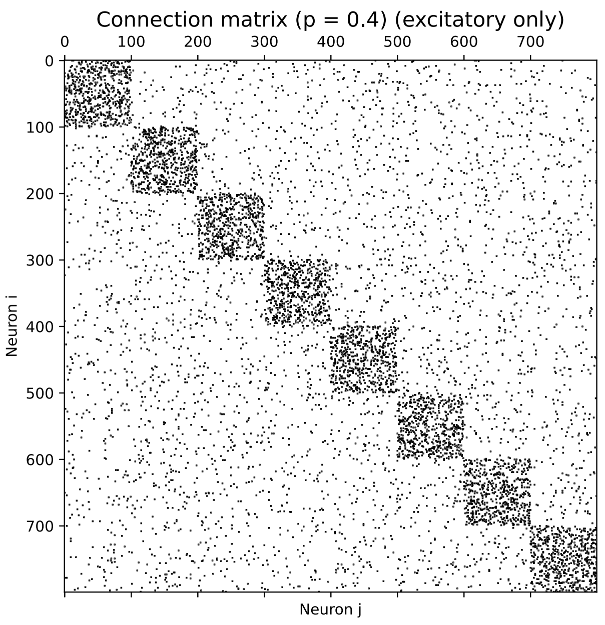
Connection Matrices

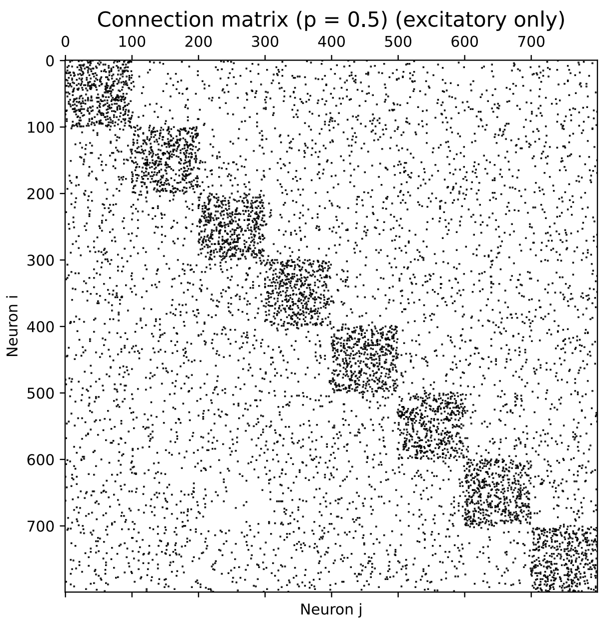




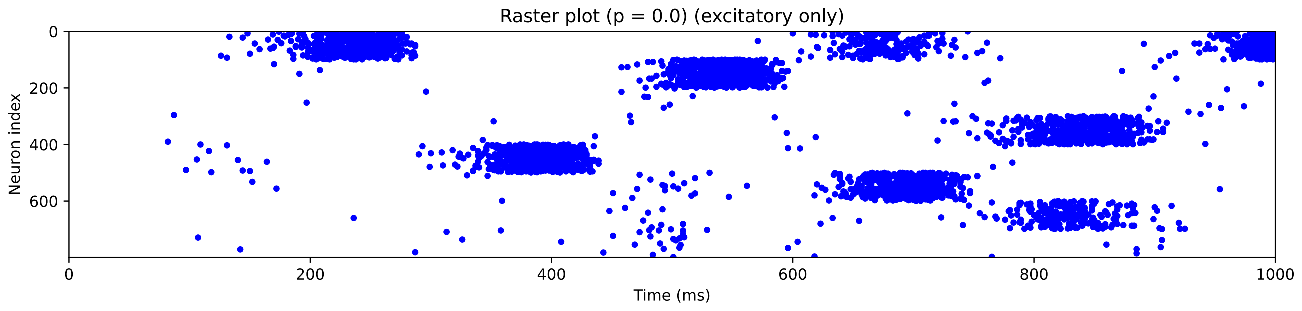


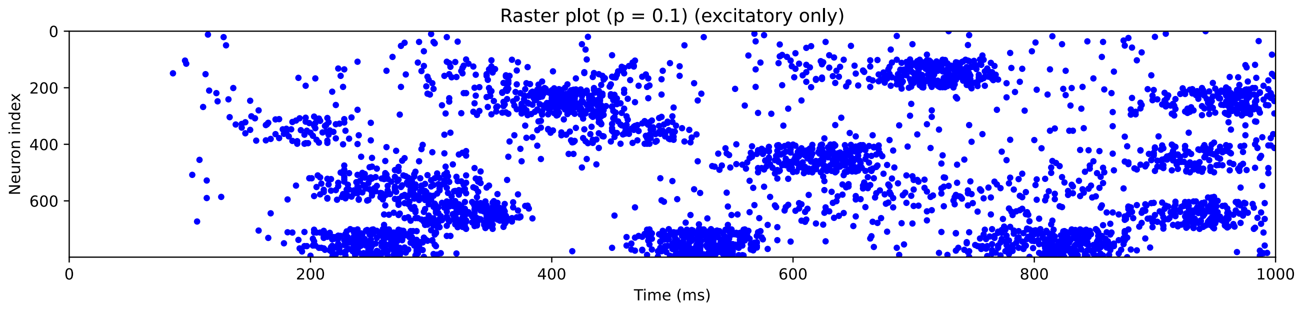


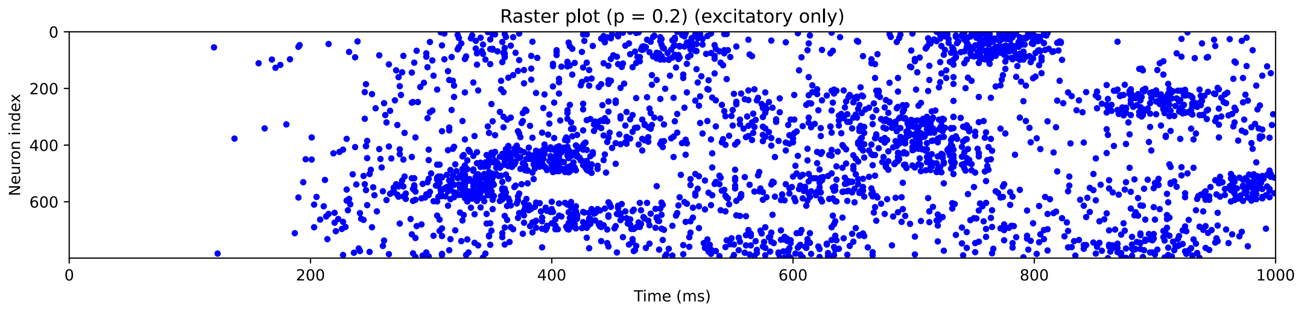


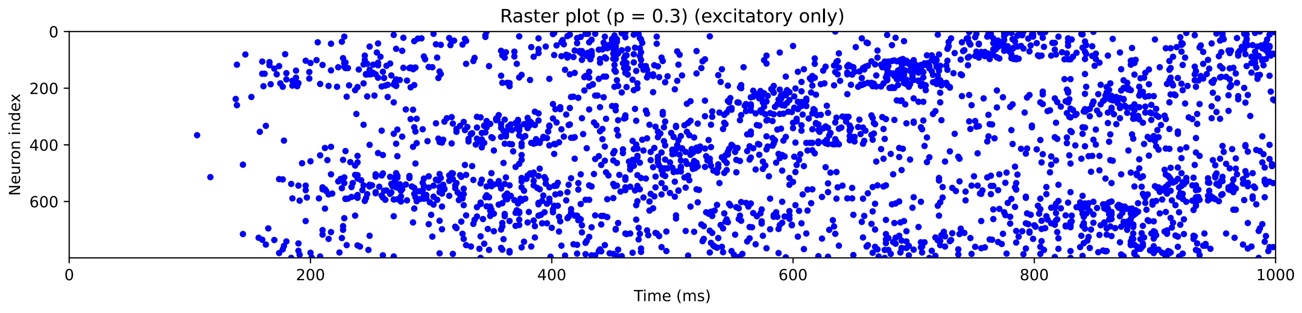


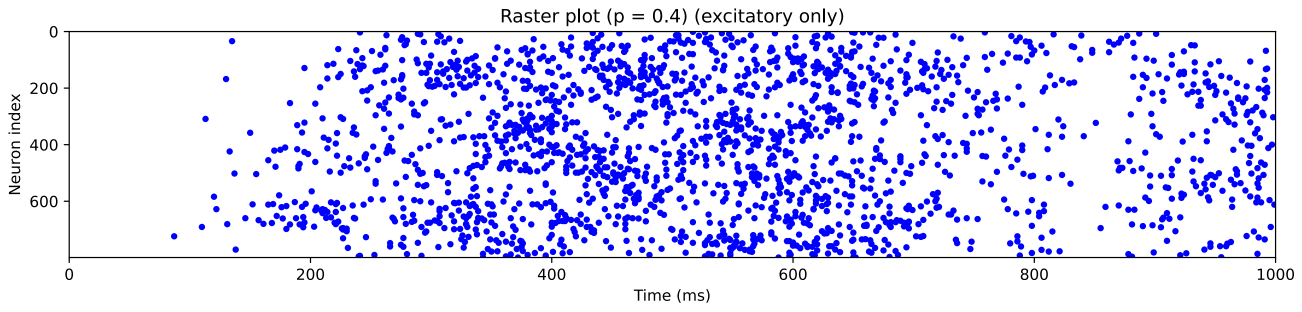
Raster Plots

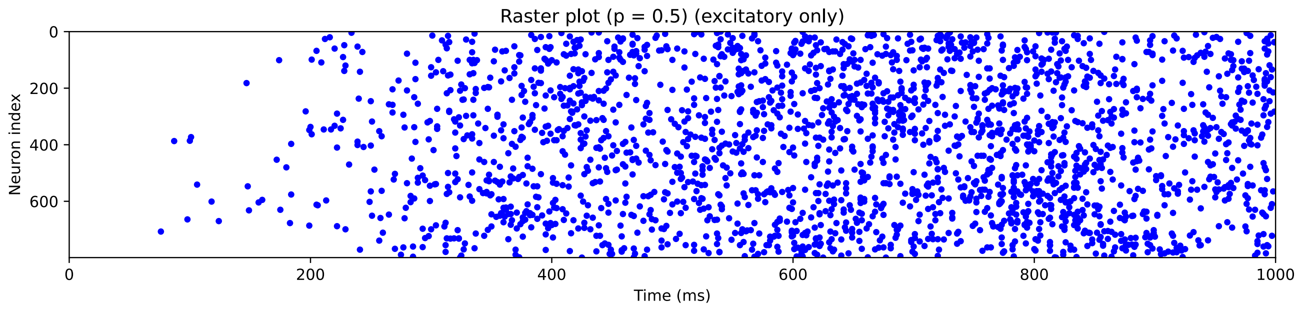




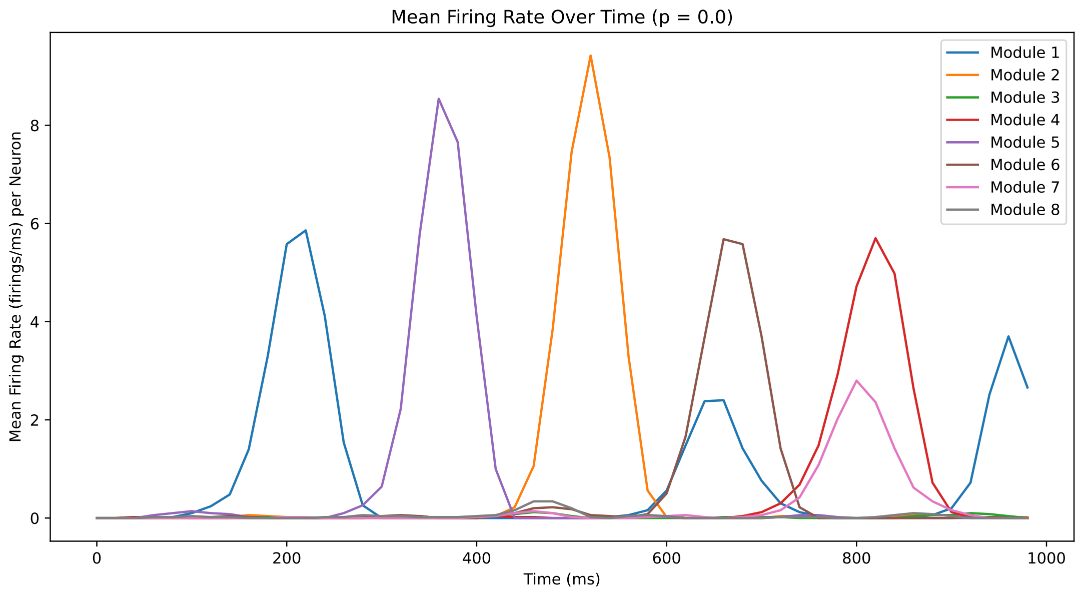


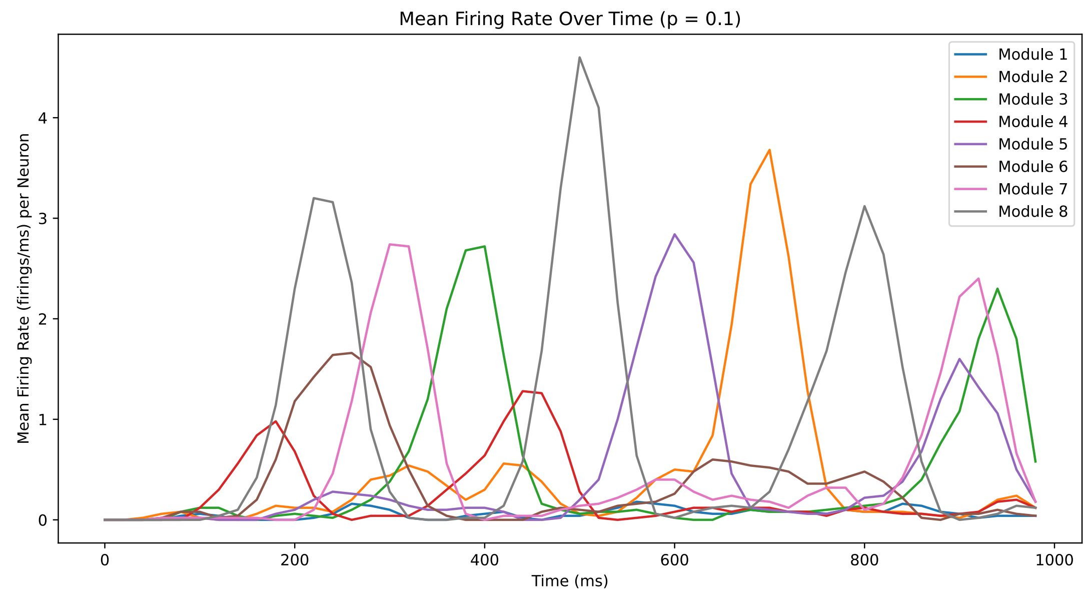


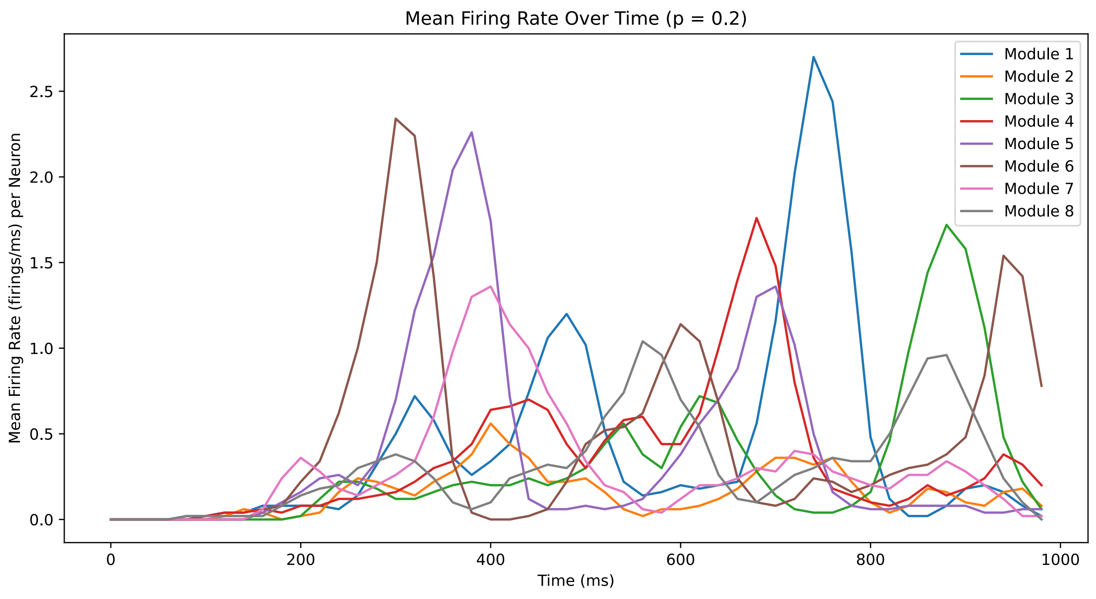


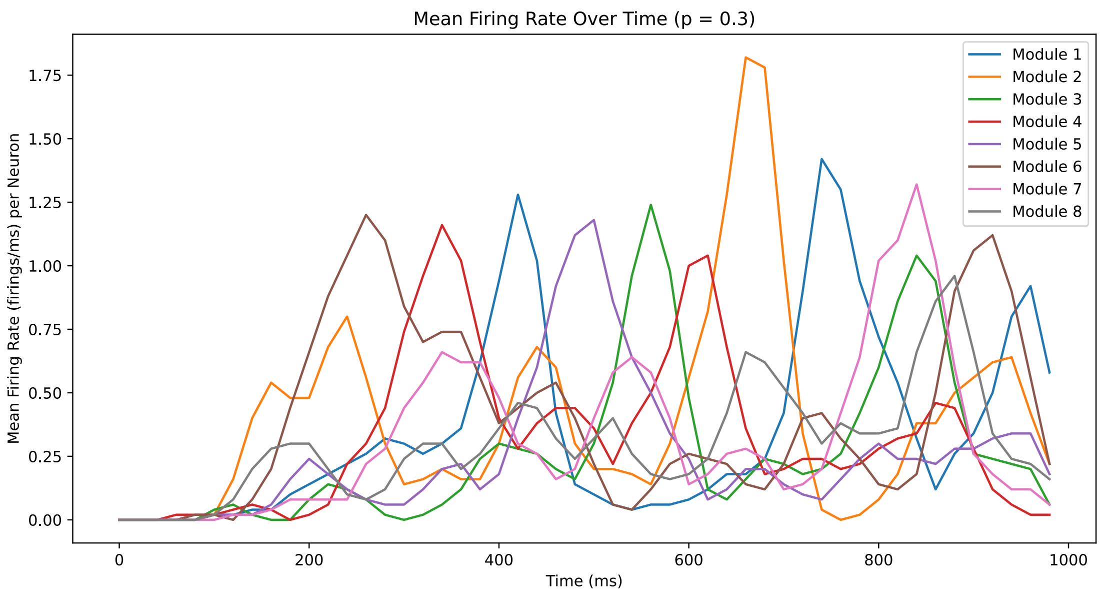


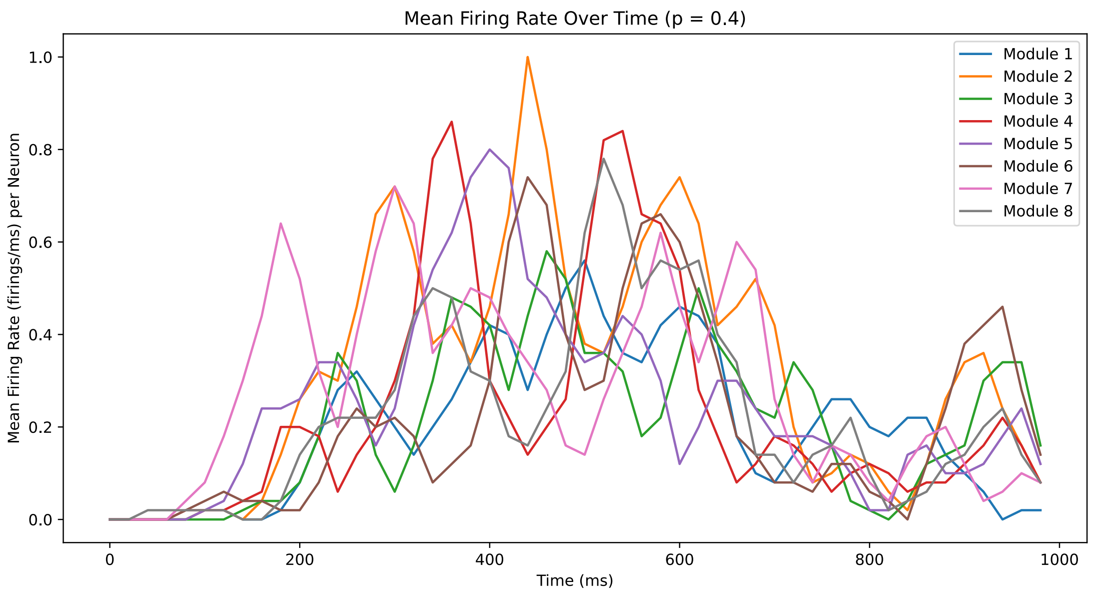
Mean Firing Rates

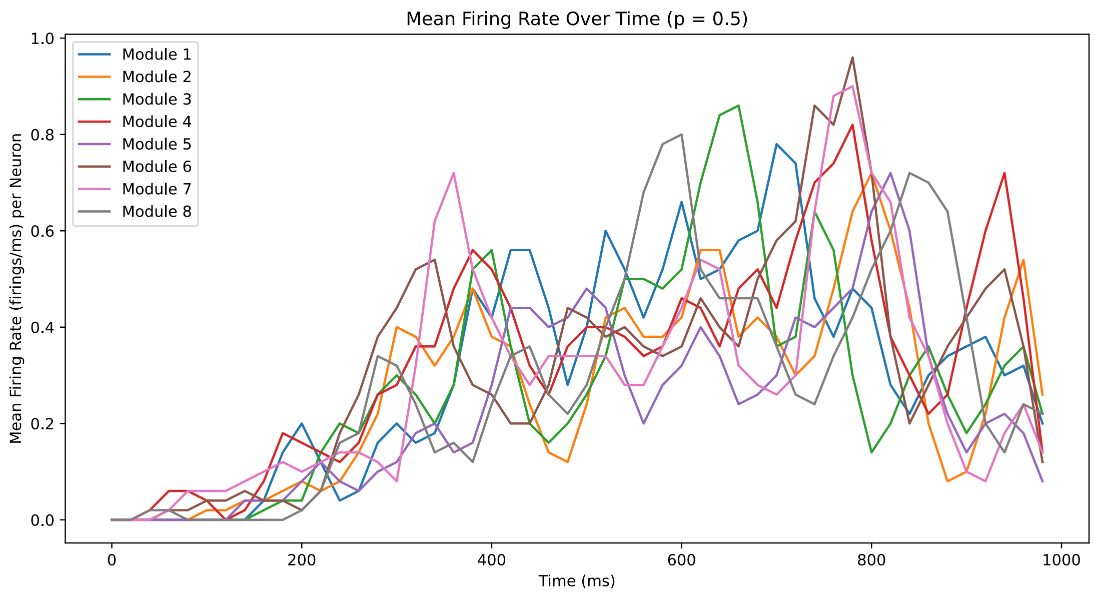



Mean Firing Rate Over Time (p = 0.1)
Mean Firing Rate (firings/ms) per Neuron
Time (ms)
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6
Module 7
Module 8


Mean Firing Rate Over Time (p = 0.2)
Mean Firing Rate (firings/ms) per Neuron
Time (ms)
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6
Module 7
Module 8

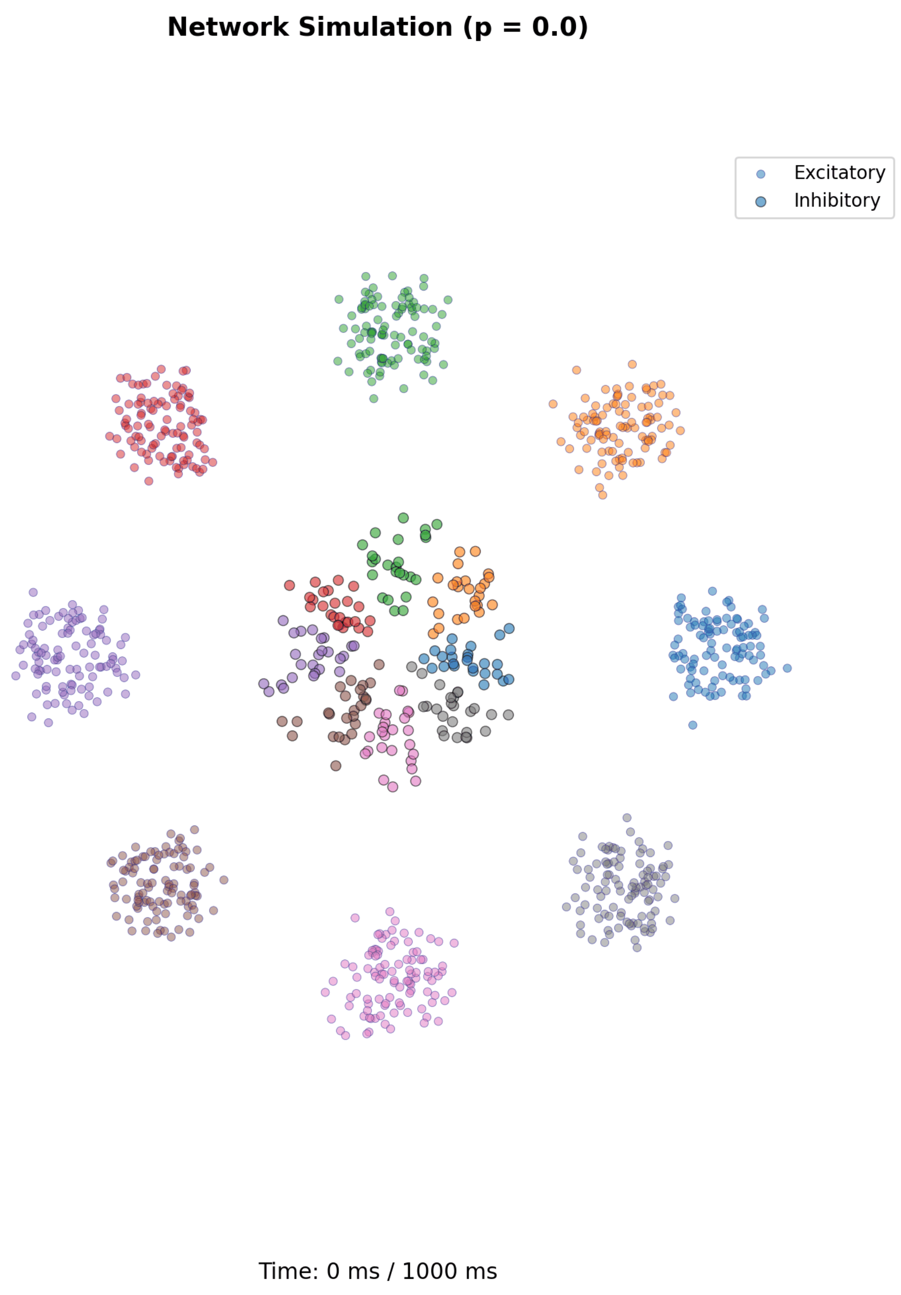

Mean Firing Rate Over Time (p = 0.3)
Mean Firing Rate (firings/ms) per Neuron
Time (ms)
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6
Module 7
Module 8

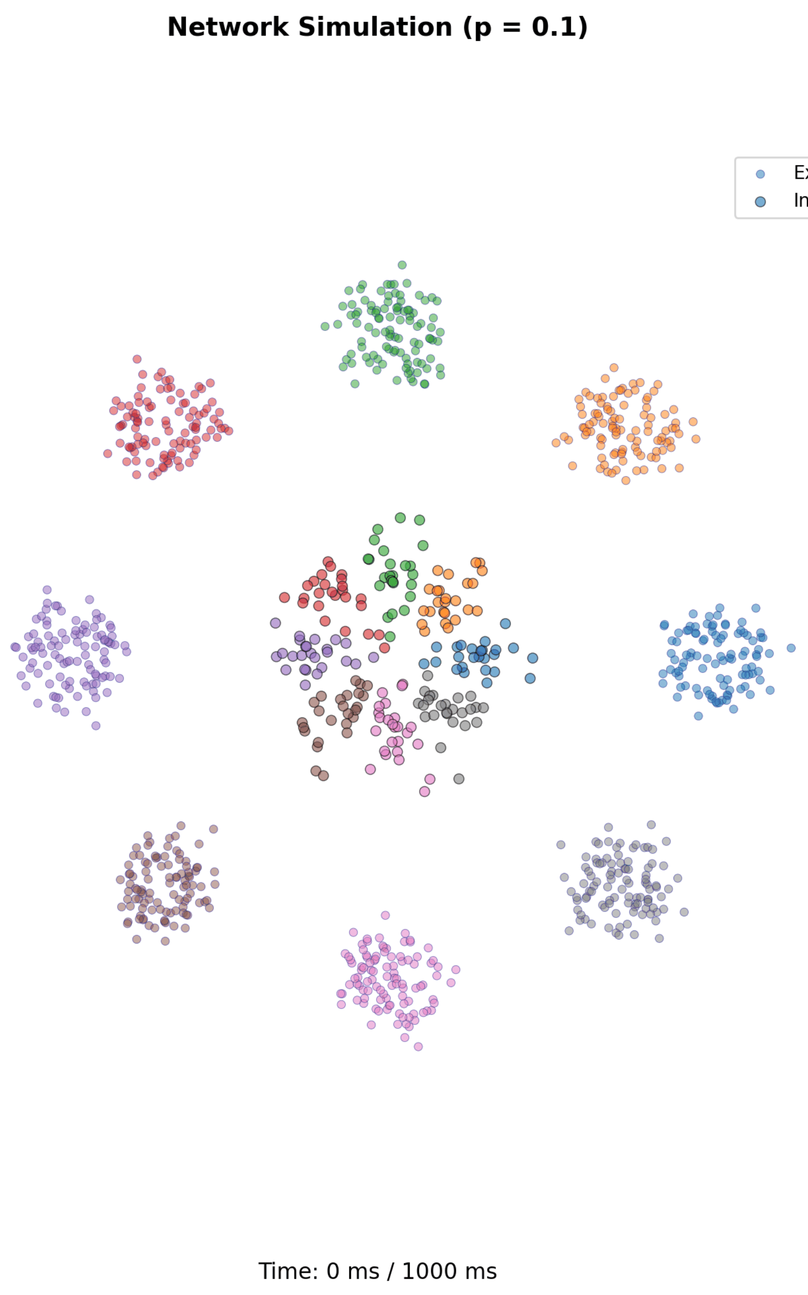

Mean Firing Rate Over Time (p = 0.4)
Mean Firing Rate (firings/ms) per Neuron
Time (ms)
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6
Module 7
Module 8

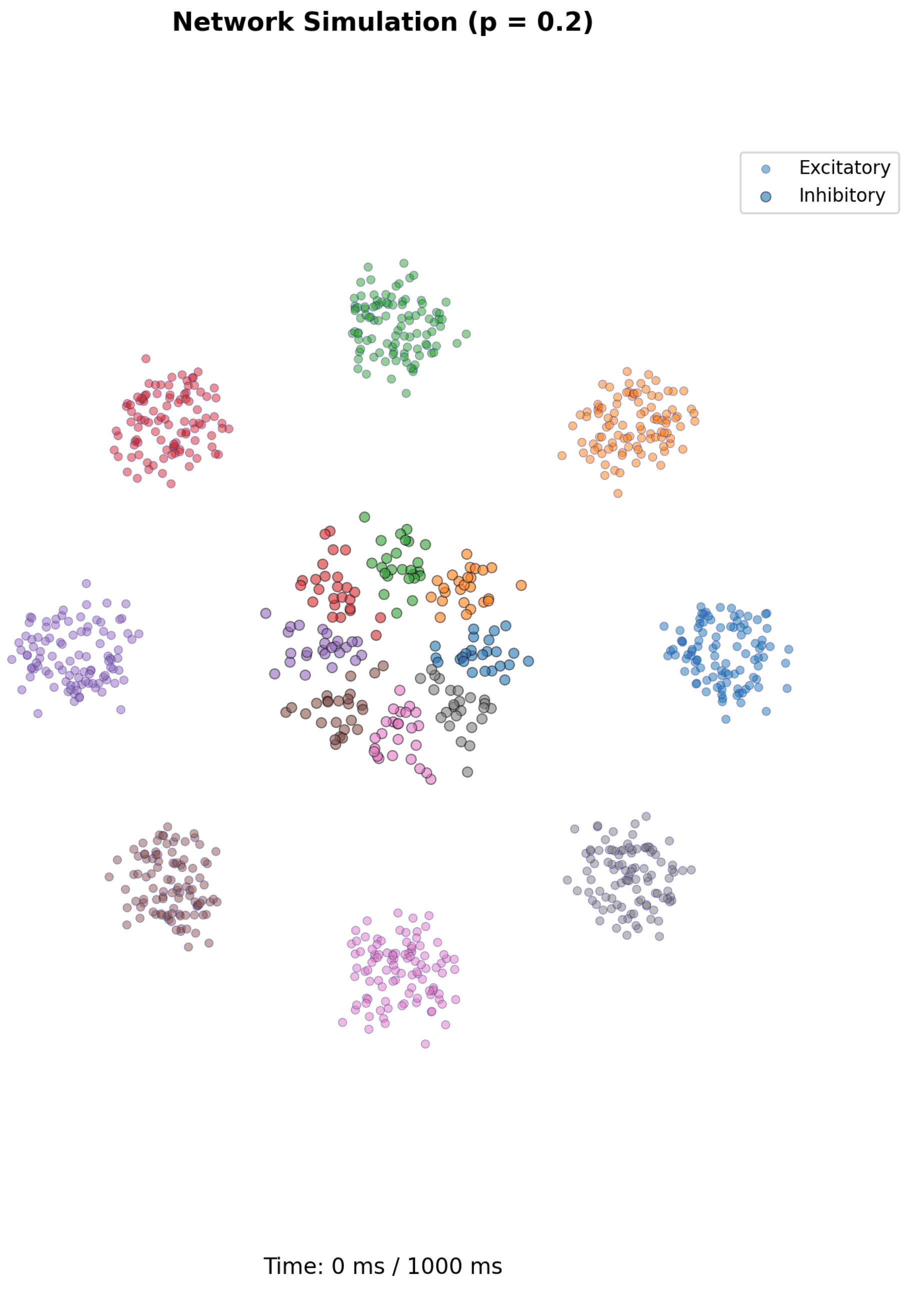


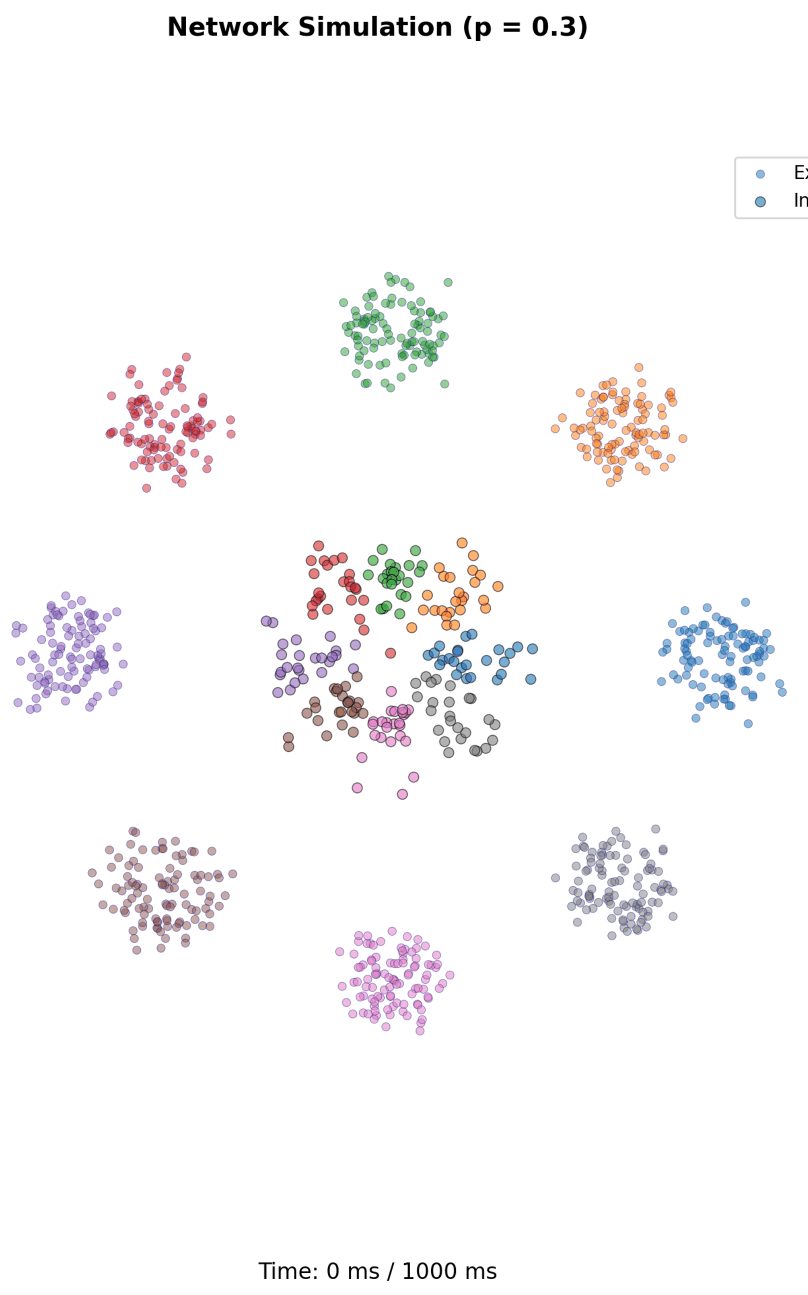
Network Visualization

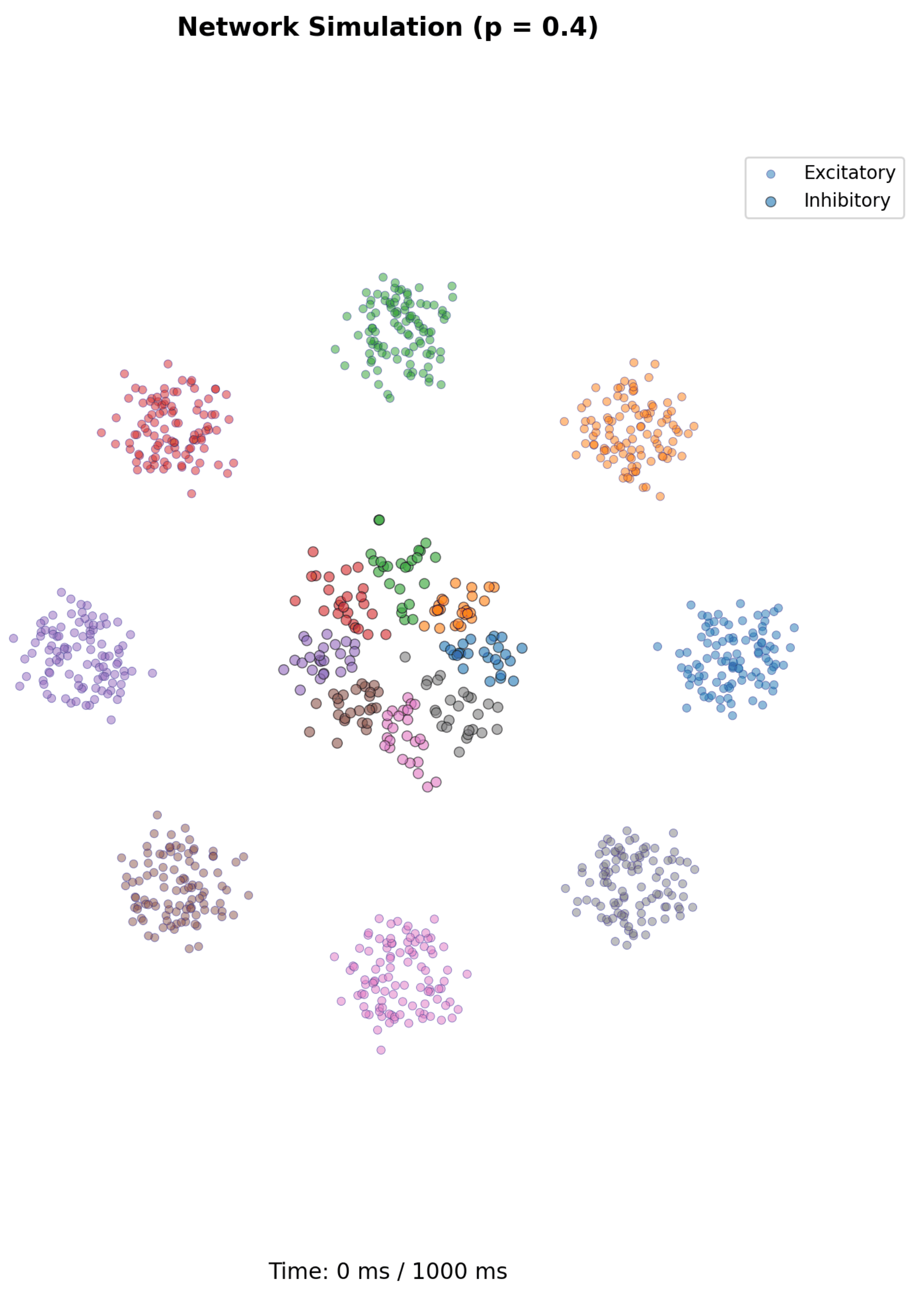
(These might take a few seconds to load)

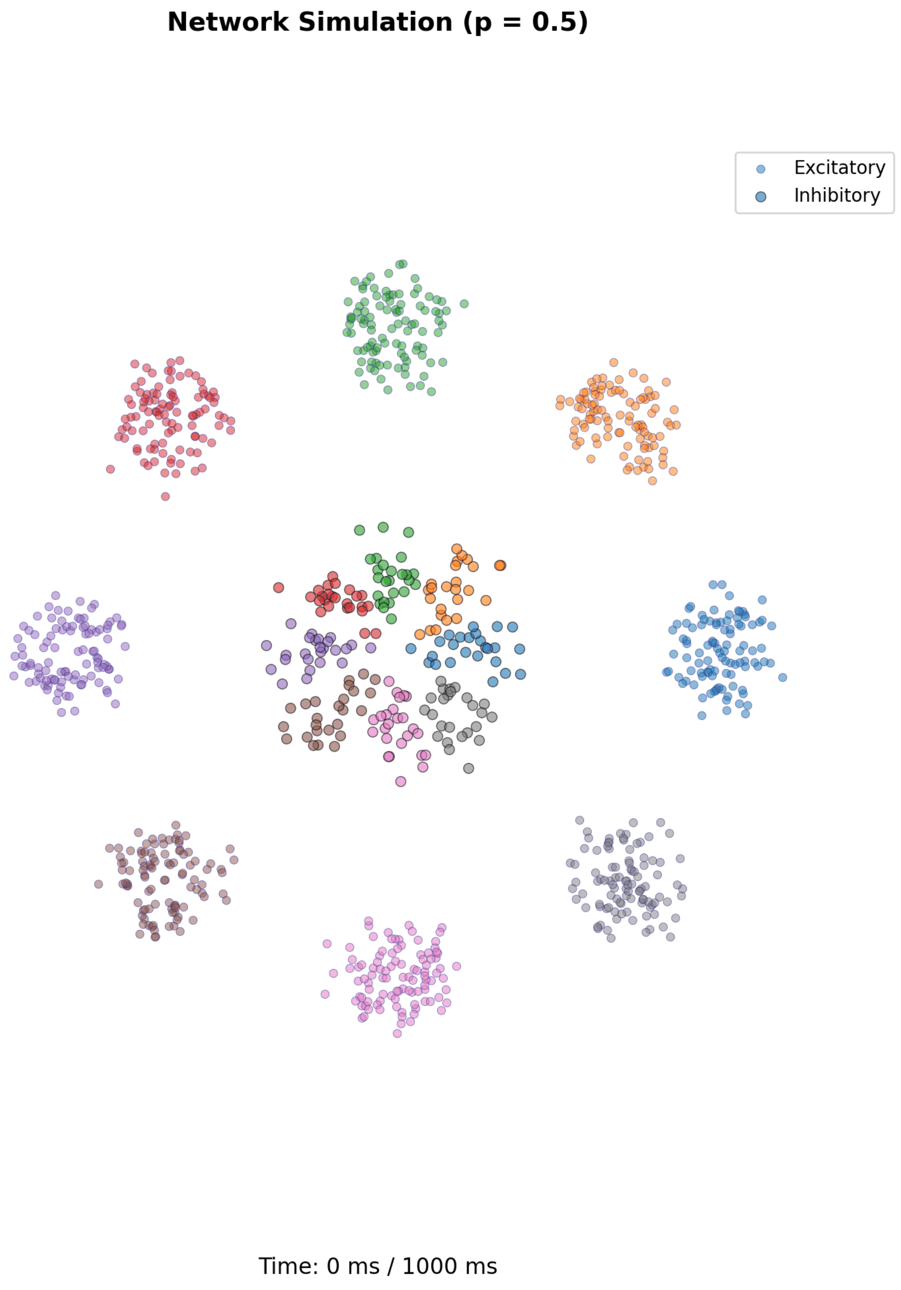

Network Simulation (p = 0.0)
Excitatory
Inhibitory
Time: 0 ms / 1000 ms


Network Simulation (p = 0.1)
Excitatory
Inhibitory
Time: 0 ms / 1000 ms


Network Simulation (p = 0.2)
Excitatory
Inhibitory
Time: 0 ms / 1000 ms


Network Simulation (p = 0.3)
Excitatory
Inhibitory
Time: 0 ms / 1000 ms


Network Simulation (p = 0.4)
Excitatory
Inhibitory
Time: 0 ms / 1000 ms


Network Simulation (p = 0.5)
Excitatory
Inhibitory
Time: 0 ms / 1000 ms