GrowNet — What This Codebase Is About

GrowNet is an AI research codebase exploring a neuron-centric, biologically inspired alternative to classic ANNs. Instead of a neuron being a single accumulating scalar with a bias, each neuron in GrowNet contains an elastic set of internal “slots” that specialize to different input regimes over time. Learning happens locally, event-by-event, without backprop, and the network can grow capacity as the data distribution shifts.

The repository provides a cross-language reference (Python, C++, Mojo; Java is the semantic “gold” reference) with a shared architecture and nearly identical public APIs. The design intentionally mirrors neuron types (excitatory, inhibitory, modulatory), lateral buses (for inhibition/modulation), region/layer wiring, and shape-aware I/O (2D today; ND path scaffolded).

Core Ideas

- Neuron as mini-system: each neuron holds a map of slot_id → Weight. A slot represents a discretized input regime (e.g., a percent-change bin). Thresholds per slot adapt to target firing rates; strengths reinforce on hits.
- Two-phase tick: a Region processes input in Phase A (inject + local routing) then flushes inter-layer tracts once in Phase B, and finally decays buses. This stabilizes feedback and keeps timing semantics clear.
- Lateral control: per-layer LateralBus carries transient inhibition and modulation, decaying each tick. Inhibitory and modulatory neurons pulse these factors rather than propagating spikes downstream.
- Temporal Focus (V4): slot selection is anchor-based (FIRST-anchor) so monotonic ramps don’t collapse into a single last-value bin. Outlier deltas can trigger growth hooks.
- Growth hooks: when a neuron is saturated (slot_limit) and observes outlier inputs, the system exposes place-holders to allocate more capacity (slots/neurons/layers) safely.

Architecture and Data Flow

- Region (`src/python/region.py`, `src/cpp/Region.*`, `src/mojo/region.mojo`)
  - Owns layers, input/output port bindings, random wiring helpers, pulses, and prune orchestration.
  - Ports as edges: binding an input lazily creates an edge layer (1 neuron for scalar; `InputLayer2D` for images) that then connects into the internal graph with probability 1.0.
  - Ticks: `tick(port, value)` for scalars; `tick_image/tick_2d(port, frame)` for images. Returns `RegionMetrics` (delivered_events, total_slots, total_synapses).
- Layer (`src/python/layer.py`, `src/cpp/Layer.*`, `src/mojo/layer.mojo`)
  - Mixed population of excitatory/inhibitory/modulatory neurons, sharing a `LateralBus`.
  - `forward(value)` drives neurons; if a neuron fired, `on_output` is called (output neurons accumulate, others usually no-op). `end_tick()` runs housekeeping and bus decay.
- Neuron (`src/python/neuron.py`, subclasses; `src/cpp/Neuron.*`; `src/mojo/neuron*.mojo`)
  - Holds slots and outgoing synapses; remembers last input; exposes `on_input(value) -> fired` and `on_output(amplitude)`.
  - Temporal Focus state: `focus_anchor`, `focus_set`, `focus_lock_until_tick` reserved for attention/locking.

    - Types: Excitatory (propagates when fired), Inhibitory (pulses inhibition), Modulatory (pulses learning-rate modulation), plus `InputNeuron` and `OutputNeuron` for shape-aware I/O layers.
- Slots, Weights, Synapses
    - Slot/Weight (`src/python/weight.py`, `src/cpp/Weight.*`, `src/mojo/weight.mojo`): keeps strength, threshold (θ), hit_count, ema_rate, and last_touched; handles reinforcement and threshold adaptation.
    - Learning rule: T0 imprint on first seeing a value ($\theta \approx |x| \cdot (1 \pm \varepsilon)$), then T2 homeostasis nudges θ toward a target spike rate via EMA.
    - Synapse (`src/python/synapse.py`, `src/cpp/Synapse.*`, `src/mojo/synapse.mojo`): directed connection; in some paths delivery is mediated by `Tract` fire hooks rather than explicit per-spike objects.
- SlotEngine and SlotConfig
    - `SlotEngine` (`src/python/slot_engine.py`, `src/cpp/SlotEngine.*`, `src/mojo/slot_engine.mojo`) computes/selects the active slot, creating it if needed.
    - Temporal Focus (V4): FIRST-anchor binning using `bin_width_pct`, `epsilon_scale`, and `slot_limit`; knobs live in `SlotConfig` (`src/python/slot_config.py`, `src/cpp/SlotConfig.h`).

Shape-Aware I/O and Ports

- InputLayer2D / OutputLayer2D (`src/python/input_layer_2d.py`, `src/python/output_layer_2d.py`; analogous C++/Mojo files).
- `Region.bind_input_2d(...)` creates or reuses a 2D input edge and wires it to attached layers; `tick_image/tick_2d` expects such an edge bound to the port.
- ND input scaffolding exists (`input_layer_nd`, `bind_input_nd`, `tickND` prototypes) and is planned for Phase B.

Metrics, Benchmarks, and Tests

- Metrics: `RegionMetrics` tracks `delivered_events`, `total_slots`, `total_synapses` with helper methods. Python: `src/python/metrics.py`. C++: struct in `Region.h`. Mojo mirrors the same surface.
- Benchmarks: `src/bench` has a language-agnostic harness and per-language templates to measure end-to-end and micro-benchmarks (e.g., slot id, reinforce, on_input).
- Tests (Python): smoke tests for single ticks, slot formation, image path, pulses, and Temporal Focus (`src/python/tests`). The Temporal Focus test asserts slot growth over a monotonic ramp.

Language Parity and Style

- Parity: Names/semantics match across Java (gold), C++, Python, and Mojo with casing differences (camelCase vs snake_case). Public methods are not removed; when behavior changes, delegating aliases are kept (e.g., `tickImage` → `tick2D`).
- Style: Python uses explicit classes (no dataclasses), snake_case methods; Mojo uses `struct` + `fn` with explicit types; C++ uses smart pointers and throws for index errors. See `docs/STYLE_AND_PARITY.md`.

How to Run (Python reference)

- Minimal region demo: `python -m src.python.demos.region_demo` (or see `docs/GrowNet_Quick_Start_for_Engineers.md`).
- Image I/O demo: `python -m src.python.demos.image_io_demo` (2D edge + output buffer). Some demo files are WIP; use `src/python/region.py`, `input_layer_2d.py`, and `output_layer_2d.py` as the reference flow.

What's New in V4 (Temporal Focus)

- Anchor-based slotting avoids "drift to last value" and enables outlier detection per neuron.
- Growth hooks are introduced (safe, no-throw) to later add neurons/layers when outliers hit slot capacity.
- Ports as edges unify scalar and 2D inputs, keeping Region decoupled from payload shapes.
- The ND path and Spatial Focus (attention over 2D/ND tensors) are specified but slated for Phase B.

See `docs/GrowNet_Design_Spec_V4.md` for the full rationale, config knobs, and cross-language signature references.

Notable Rough Edges (WIP Notes)

- Some Python demos contain minor naming mismatches (e.g., method casing/typos); the reference types under `src/python/` reflect the intended API.
- Inhibitory/Modulatory fire hooks differ slightly in naming versus the base (`fire_hooks` vs `_fire_hooks`) and may need alignment in a cleanup pass.
- Region-wide bus is optional in Python (`Region.bus = None`); per-layer `LateralBus` is the primary control path today.

Where to Look Next

- Design and API: `docs/GrowNet_Design_Spec_V4.md`, `docs/GrowNet_API_One_Pager.md`, `docs/GrowNet_Field_Guide_v3.md`.
- Quick start and tutorial: `docs/GrowNet_Quick_Start_for_Engineers.md`, `docs/GrowNet_Tutorial.md`.
- Cross-language code: `src/python/*`, `src/cpp/*`, `src/mojo/*`.
- Benchmarks and templates: `src/bench/*`.

In short, GrowNet is a unified, event-driven neural substrate with per-neuron slot structure, transient lateral control, and explicit I/O edges. The codebase provides aligned implementations across languages and a path toward growth and spatial focus that we intend to evaluate and present (e.g., NeurIPS) as the research matures.