GrowNet, in Plain Language

A friendly guide for a non-technical reader (and a brief note about NeurIPS)

---

1) The one-paragraph version

GrowNet is a research project about building learning systems that behave a little more like simple biological circuits and a little less like today's large, math-heavy deep-learning pipelines. Instead of nudging millions of numbers with backpropagation, GrowNet's "neurons" each keep a small set of **memory slots** that specialize to patterns they see—like labeled drawers that fill up with familiar experiences. Learning is **local and event-by-event**: when something happens, the neuron updates its own slots and decides whether to "fire," and the system moves on. Over time the network can **grow**, adding new slots (and later neurons/layers) if the world changes. It runs in multiple languages (Python, C++, Mojo, Java) with the same basic design so it's easy to test and compare.

---

2) Why we're building it

- **To learn continuously, not just in big training runs.** Most deep nets are trained in large batches and then deployed. GrowNet aims to learn on the fly—**tick by tick**—so it can adapt as data shifts.
- **To keep decisions understandable.** Each neuron's behavior is tied to a handful of **named slots** (simple bins), which makes it more transparent why a neuron fired.
- **To explore biology-inspired ingredients**—like **excitatory/inhibitory** populations, a **lateral bus** that temporarily boosts or dampens activity, and **homeostatic thresholds**—in a compact, programmable framework.
- **To try a different trade-off space.** This is not a replacement for deep learning; it's a **complementary path** that may be better for small/on-device settings, quick adaptation, or where interpretability matters.

---

3) What makes GrowNet different

1. **Neuron with slots, not just one number.**
   In most models, a neuron is "sum inputs → apply a nonlinearity." In GrowNet, a neuron has **an elastic set of slots**—think of them as **buckets** for different kinds of inputs (for example, small change vs. large change). The neuron learns which bucket to use and how strong its response should be. No global gradient is needed; it's a **local rule**.
2. **Two-phase timing.**
   Each input tick proceeds in two steps: (A) inject and do local work, (B) flush inter-layer signals, then **decay** the temporary boosts/suppressions on the **lateral bus**. This clean timing keeps feedback loops stable.
3. **Ports as edges.**
   Inputs (numbers or images) connect through **explicit "edge" layers**. A tick drives the bound edge **once**, and wiring fans that activity into the network. This keeps the core simple and makes image/ND inputs "feel" the same as scalars.

4. **Temporal Focus (today) and Spatial Focus (in progress).**
    - **Temporal Focus**: neurons **anchor** to an early reference and then bucket future changes by **percent change** relative to that anchor (stops everything collapsing into "whatever I saw last").
    - **Spatial Focus** (2D/ND): neurons can form **location bins** (row/col buckets) so the system can pay coarse attention to *where* a signal is on an image. This opens the door to motion, centroids, and simple shape cues without heavy CNN machinery.
5. **Cross-language by design.**
   The same architecture exists in **Python, C++, Mojo, and Java** so we can cross-check results and optimize for different environments. The Java version is the "semantic gold" reference; the Python version is often the friendliest to read and test.

---

4) How to picture it (everyday analogies)

- **A spice rack with labeled jars.**
  Each neuron is like a cook's spice rack. The first time a new flavor shows up, the cook creates a new jar and labels it ("mild," "medium," "smoky"). As similar flavors come back, the cook reaches for the matching jar faster. When the pantry gets crowded, the cook can stop adding jars or "freeze" a favorite jar to keep it from changing.
- **A city traffic system with signals.**
  The **lateral bus** is like the short-lived "rush hour" policies: a bit of inhibition (red lights) or modulation (green-wave timing) that fades after each cycle. It helps prevent everyone from honking at once (runaway activity) and keeps the flow smooth.
- **A moving dot on a screen.**
  With **Spatial Focus**, nearby pixels get binned together; as a dot moves **right → left**, different spatial bins light up in sequence. The pattern of bins (and how quickly they change) is a simple hint about motion and speed—no big convolution stack needed.

---

5) How neuroscience ideas show up

- **Excitatory / Inhibitory / Modulatory neurons.**
  These three roles loosely echo real circuits: excite to pass information, inhibit to prevent overload or sharpen signals, and modulate to change how quickly we learn.
- **Homeostasis (keeping balance).**
  Each slot has a **threshold** that drifts toward a healthy firing rate, so activity neither dies out nor explodes. This is inspired by **homeostatic plasticity** ideas from neuroscience.
- **Receptive fields and attention.**
  Temporal/Spatial Focus are the system's simple way of forming **receptive fields**—not full vision cortex models, just lightweight bins that say "I pay attention to *this kind* of change" or "to *this place* on the image."

---

6) What might this benefit in AI?

- **Continual learning.**

Because learning is local and on-line, GrowNet can adapt without staging a full retrain. That's appealing for **embedded** or **edge** uses.

- **Small-data or streaming scenarios.**
  Slotting is naturally **data-efficient** for stable patterns—it's like building a tiny histogram of what matters for each neuron.
- **Interpretability.**
  You can inspect a neuron's slots (their thresholds, strengths, and hit counts) to understand *why* it fired. That's far easier than peering into a giant weight matrix.
- **A new testbed for neuro-AI ideas.**
  The architecture is compact enough to try out **growth**, **attention**, and **stability** ideas quickly—and in multiple languages—while still being principled.

---

7) Where this goes next (and what's "unique")

- **From slots to growth.**
  The codebase already includes **growth hooks**—signals that say "I'm full and seeing new stuff." That enables controlled expansion of slots, then neurons, then (eventually) layers/tracts. It's a **bottom-up growth** story rather than a fixed, pre-sized model.
- **Phase B: Spatial Focus.**
  We've added 2D anchoring, windowed wiring, and optional spatial metrics (like centroid and bounding box) so the system can talk about **where** activity is. This keeps the framework light while opening motion/detection possibilities.
- **Cross-language parity.**
  Most research repos pick one language. GrowNet keeps a common API in Python/C++/Mojo/Java so ideas are **portable** (research), **fast** (C++), and **deployable** (Java/embedded).
- **Eight-year idea becoming code.**
  This project distills an **eight-year mental experiment** into a runnable platform, making it possible to test the hunch: *local, slot-based learning can be simple, stable, and useful.*

---

8) What NeurIPS is (and why aim there)

NeurIPS (Conference on Neural Information Processing Systems) is one of the world's top research venues for machine learning and computational neuroscience. It's where **new learning principles**, **algorithms**, and **neuro-inspired models** are proposed, debated, and benchmarked.

We're aiming at NeurIPS because GrowNet sits **right at the intersection** of ML and neuroscience: it proposes a **concrete, testable alternative** to global gradient training, it's **interpretable**, and it's built to enable **ablation studies** (what happens if we change slot thresholds, bus decay, spatial bins, etc.) across **multiple runtimes**. That makes it a natural fit for both the theory and systems tracks.

---

9) What exists today (and how to try it)

- **Working reference** with common APIs in Python, C++, Mojo, and Java.
- **Temporal Focus** is implemented; **Spatial Focus** features and metrics are available in Python (with C++

support for some parts).
- **Metrics** help you see how many events were delivered and how many slots/synapses exist after each tick.
- **Demos and tests** show image ticks (2D), scalar ticks, and slot growth on a simple ramp.

If you're curious but not technical, think: *"When a small dot moves across a grid, the system notices **which small neighborhoods** light up and **how quickly**. Those patterns get their own labeled drawers (slots). Later, when the dot returns, the system recognizes it faster and reacts more confidently."*

---

10) How this differs from "classic deep learning"

- **No backprop.** GrowNet doesn't compute global gradients. Each neuron updates itself based solely on what it just saw.
- **Elastic capacity.** Instead of fixing size up front, neurons can **add slots** as needed (and later we can grow the graph).
- **Clear timing and control.** The **two-phase tick** and **lateral bus** make it explicit when signals move and how instability is avoided.
- **Explainability.** You can point to the **specific slot** that fired and see its threshold and reinforcement history.

---

11) What to watch for next

- **Benchmarks** comparing slot-based learning to small neural baselines on streaming tasks.
- **Growth experiments** (when to add slots or neurons) and **frozen slots** (locking in a useful feature).
- **Richer spatial behavior** (e.g., motion cues, simple object persistence) without large convolution stacks.
- **Energy/footprint profiling** to see where this approach shines (e.g., on-device learning).

---

12) Closing thought

This is a **research journey**, not a product. The goal isn't to "beat" deep learning at everything. It's to **open another lane**: small, interpretable, locally learning networks that can grow with their environment. After almost eight years of sketching and refining the idea, GrowNet is now a living codebase where we—and the community—can test it for real.

If you'd like a two-minute demo walkthrough (no code), I can summarize a "moving dot" example with what the slots learn and how the spatial metrics (centroid, bounding box) change with each tick.