

SoME v1 "Protostar": Integrating LVQ/Hebbian dynamics inside a Transformer MoE

What is a Mixture of Experts (MoE): A Mixture of Experts is a neural-network layer that contains many sub-networks called experts (often small feed-forward nets). A separate router/gating network looks at each input (in LLMs: each token) and activates only a small subset of experts (e.g., top-1 or top-2). Their outputs are combined (usually a weighted sum) and passed on. You get huge parameter count but sparse activation, so per-token compute stays roughly constant while capacity (total params) can skyrocket. In modern Transformers, MoE typically replaces some FFN blocks: the router scores all experts, selects top-k, dispatches tokens to those experts, combines results, and adds regularizers to keep routing balanced (so no single expert hogs the traffic).

So what is Self-Organizing Mixture of Experts (SoME): The idea of SoME is to Integrate LVQ/Hebbian dynamics inside of a Transformer MoE architecture. We'll achieve these with the following.

1. The Core Innovation: Decoupling Function from Address, This is the central, non-negotiable principle of SOME.

- Static Function (The "What"): Each expert's core knowledge, its internal weights, is frozen after pre-training. These experts are reliable, stable specialists. This directly solves the problem of catastrophic forgetting, as the knowledge base itself is never corrupted during inference.
- Dynamic Address (The "Where"): Each expert is assigned a "routing key"—a small, low-dimensional vector that represents its address in a high-dimensional conceptual space. These keys are plastic and are continuously updated during inference.

The model doesn't re-learn how to do things; it re-learns where to find the expert that knows how to do things. It's a self-organizing "address book" for its own knowledge.

2. The Self-Organizing Mechanism: "Knowledge Gravity", This is the "how" of the architecture. Instead of a trainable router network, SOME uses a set of gradient-free, heuristic update rules that govern the movement of the dynamic keys. We've formalized these metaphorical forces into a clear, dual-update mechanism.

Attractive Forces (Consolidation):

- Query Pull (Relevance Attraction): An expert's key is pulled toward the centroid of the query vectors that activate it. This is analogous to a k-means update and ensures that experts drift toward the "conceptual space" of the problems they are good at solving.
- Peer Pull (Hebbian Co-activation): The keys of experts that are frequently activated together for the same query are pulled closer to each other. This is the explicit "gravity" that forms "knowledge galaxies" or self-organizing neighborhoods of complementary experts (e.g., a "Python basics" cluster).

Stabilizing Forces (Equilibrium): A system with only attractive forces will collapse. This framework wisely introduces two countervailing forces:

- Usage Inertia ("Gravitational Mass"): The learning rates for updating an expert's key are scaled down by its activation frequency. Popular, generalist experts thus have higher

inertia, making them stable "galactic centers" and preventing them from being erratically moved by niche queries.

- Repulsive Decay ("Dark Energy"): To prevent all keys from collapsing into a single point, a repulsive force is introduced. The most robust implementation as noted is a Decay to Origin mechanism, where the keys of infrequently used experts slowly drift back toward the origin, effectively "pruning" or recycling them.

3. Query Fall: An incoming query doesn't get sent to an expert; it falls toward a region of knowledge. (1) The Standard MoE/MoME Paradigm (The Switchboard Operator), In a typical MoE, the router acts like a switchboard operator or a classifier. It looks at an incoming query (q) and makes a deliberate, discrete decision: "Based on my training, I will connect this query to expert #5 and expert #8." The MoME's hierarchical router, while more sophisticated, is still a series of these discrete decisions: "I'll send it to the 'Science' group, and within that, the 'Physics' sub-group, and from there, to expert #45,982." The relationship is Query \rightarrow Decision \rightarrow Expert. The router is a separate entity that maps inputs to a set of pre-defined slots. This is why it can become static and fail to adapt. (2) The SOME Paradigm (Knowledge Gravity) aims to address this, SOME removes the "decision-maker" from the middle. The query vector q and the expert address keys k_i exist in the same high-dimensional semantic space. Routing is not a decision; it is an emergent property of proximity. The query itself exerts a "gravitational pull" in this space. The experts whose keys are closest are the ones that "feel" this pull the strongest and are thus activated. The relationship is Query \rightarrow Attraction \rightarrow Expert Cluster. The system performs a massive-scale similarity search, finding the keys that "resonate" most strongly with the query. This shift from a deliberative router to an emergent, gravity-based system is what unlocks the core benefits you've designed:

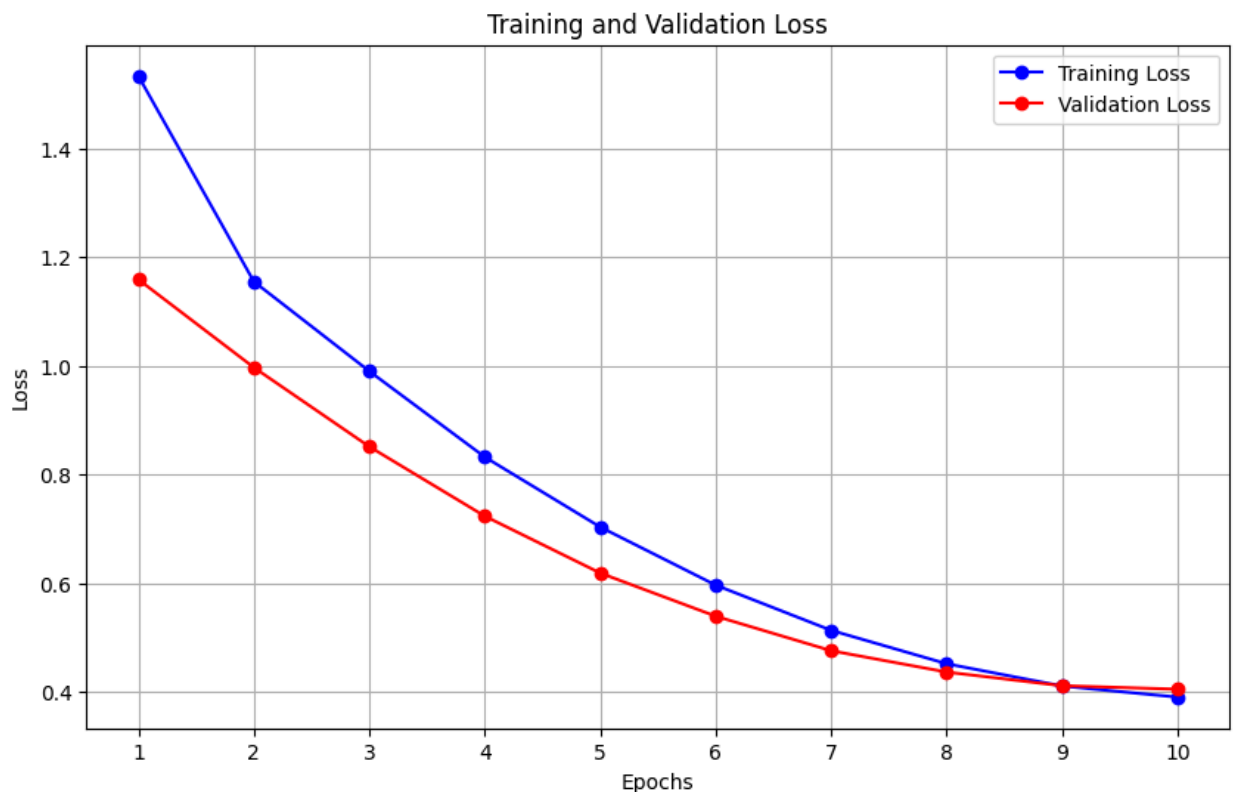
- True Self-Organization: The "address book" of experts isn't just updatable; its very structure is fluid. The "Peer Pull" and "Query Pull" mechanisms are constantly warping this semantic space, moving experts, and forming and reforming "knowledge galaxies." A static router cannot do this. Because the query simply "falls" to the most relevant region, it automatically benefits from this continuous reorganization.
- Overcoming the Static Router: The "static router" problem is completely sidestepped. There is no static router to become obsolete. As new data patterns emerge, the dynamic keys of the relevant experts are pulled into new positions. The next time a similar query arrives, it will naturally "fall" toward this newly formed cluster, achieving in-inference adaptation without any gradient-based updates to a routing network.
- Enhanced Scalability and Efficiency: The "gravity router" (implemented via massive-scale similarity search) is fundamentally more scalable. Instead of training a classifier with a million output heads (which is intractable), we are pre-building an efficient search index (like HNSW, as noted) over the key space. This is a lookup problem, not a classification problem, which is orders of magnitude more efficient at extreme scales.
- Conceptual Integrity: The "gravity" metaphor holds true throughout the system.
 - Query Pull: A query's gravity pulls relevant experts toward it.
 - Peer Pull: Co-activated experts exert gravity on each other, forming clusters.

- Usage Inertia: Frequently used "generalist" experts have more "mass," making them harder to move—they become the stable centers of knowledge galaxies.
- Repulsive Decay: "Dark energy" prevents the total gravitational collapse of the entire knowledge universe into a single point.

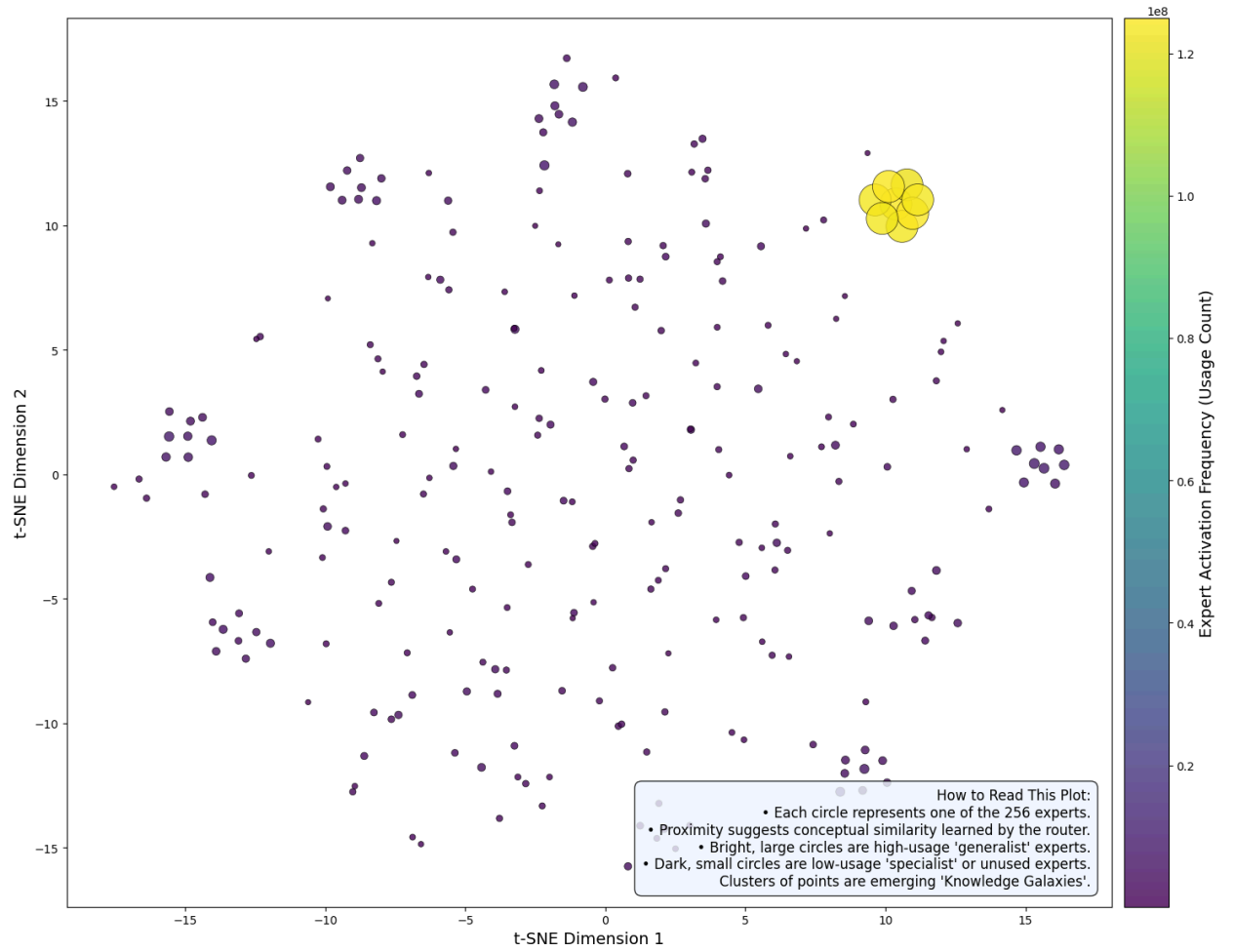
What we ran & got:

- Core: d_model=512, num_layers=8, num_heads=8
- MoE: num_experts=256, top_k=8, d_ffn=1536, alpha=0.01, beta=0.001, delta=0.001, theta_percentile: 0.05, warmup_steps: 2000
- Data: seq_len=512, batch_size=64, vocab_size=8192
- Train sizes: 40k, Val sizes: 10k
- Train: epochs=10, lr=6e-4, AdamW (wd=0.1), AMP + torch.compile
- Size: Total Parameters ~3.24B, Trainable ~18.9M
- GPU usage: 47.6 GB of 80 GB available
- Results (by first epoch): Train Loss = 1.5334, Val Loss = 1.1589, Val Perplexity = 3.19
- Results (by last epoch): Train Loss = 0.3904, Val Loss = 0.4049, Val Perplexity = 1.50

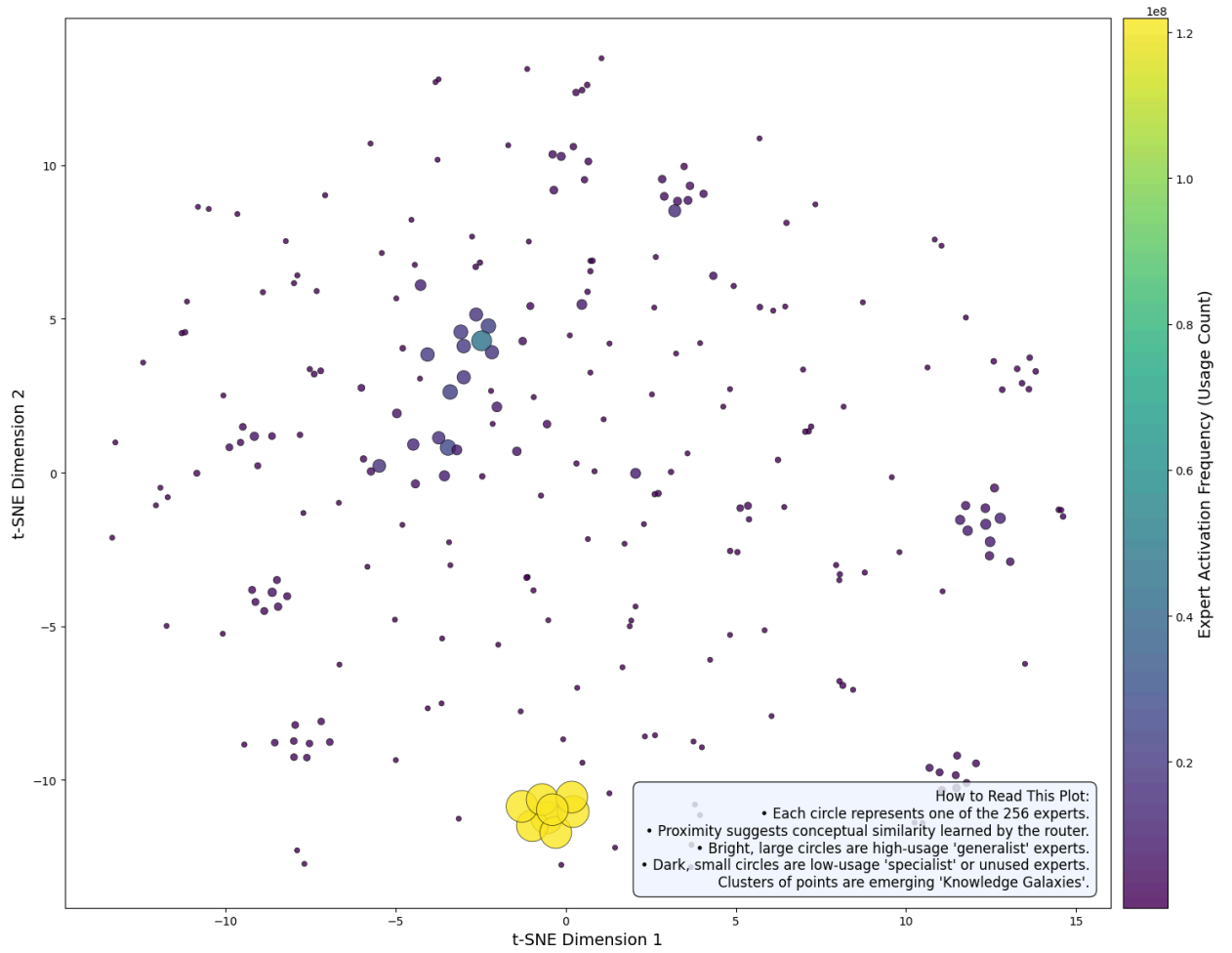
Image Results of First Run (To scale):

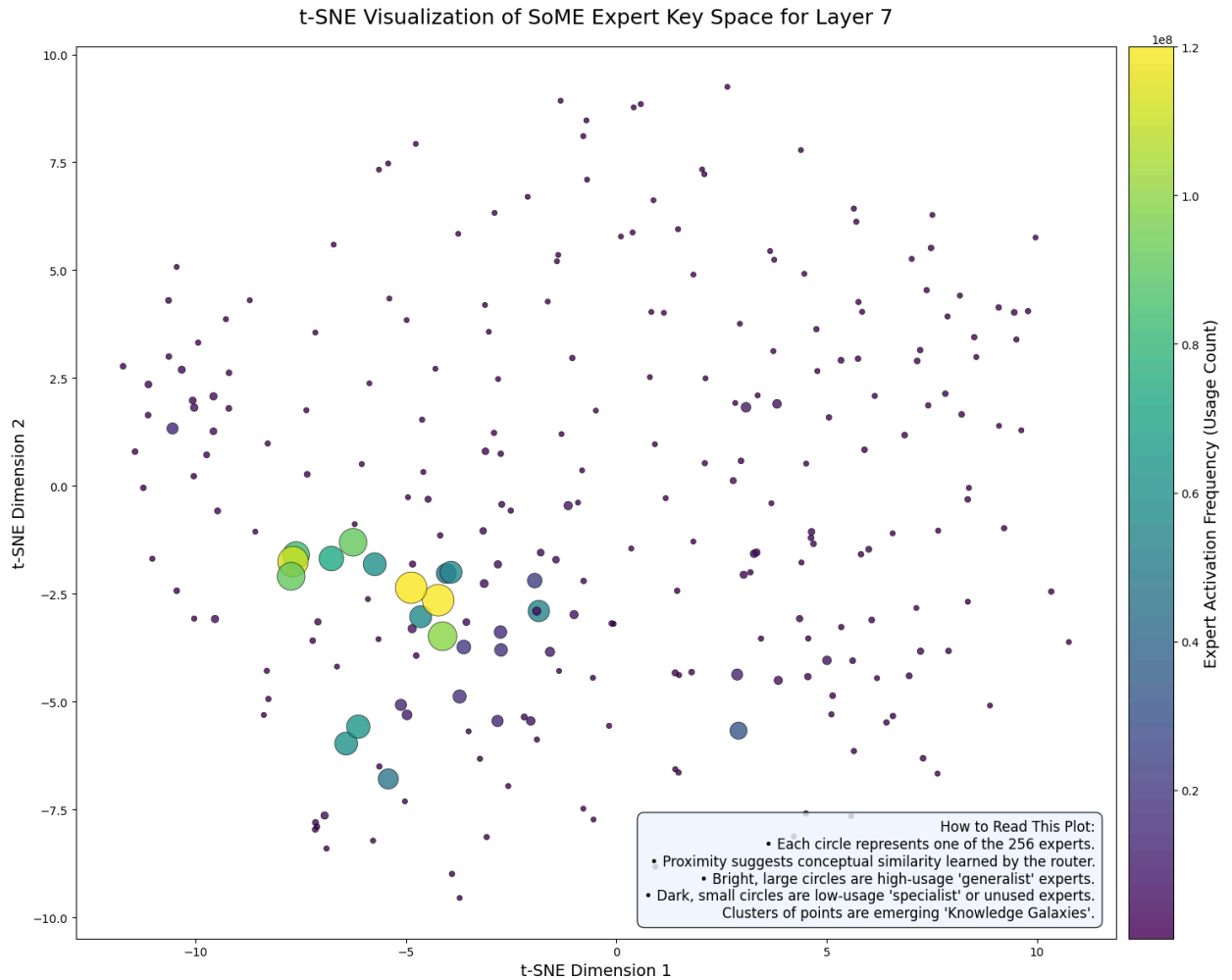


t-SNE Visualization of SoME Expert Key Space for Layer 0



t-SNE Visualization of SoME Expert Key Space for Layer 4





What's in the prototype (quick audit):

- SOMELayer: top-k by dot-product(q , key), softmax gating, frozen MLP experts, unit-norm keys, query/peer updates with inertia, percentile-based decay after warmup.
- Training loop: updates keys every step (post-optimizer), logs Gini/entropy of usage, plus a t-SNE "galaxy" viz cell and a minimal interactive generator.
- Design stance: experts are intentionally not trained; the "intelligence" emerges from selecting and combining fixed tools, consistent with the doc's "workshop of weird tools."

What's next:

1. Causal masking is missing in attention (critical for AR LM):

In SOMETransformerBlock.forward, attention is called as:

- `attn_output, _ = self.attention(x, x, x)`

with no mask. For next-token prediction you must use a causal mask (or `is_causal=True` in recent PyTorch). Otherwise the model can “peek” at future tokens during training, producing misleadingly low loss and poor generation.

Fix:

- If on PyTorch ≥ 2.0 : `attn_output, _ = self.attention(x, x, x, is_causal=True)`
- Otherwise: create an upper-triangular mask of shape `(seq_len, seq_len)` with `-inf` above the diagonal and pass as `attn_mask`.

2. EOS handling during training

- We define an EOS token and use it in generation, but the training pipeline doesn’t explicitly append EOS to each sequence. With `padding="max_length"`, many samples may never contain EOS, reducing the usefulness of EOS-based stopping and potentially harming generation quality.
- Fix: either append EOS before padding (truncate to leave room) or accept that EOS will be rare and adjust stopping accordingly.

Gaps & concrete improvements -

1. Make routing scale-free (cosine) and temperature-controlled: Right now keys are L2-normalized, queries aren’t. Query magnitude can dominate gating, So normalize queries and add a temperature schedule so early training explores, later sharpens.

2. Usage inertia should be adaptive, not cumulative: Cumulative counts will “freeze” early popular experts forever. Use an Exponential Moving Average (EMA) window so inertia reflects recent traffic. This should preserve plasticity while keeping stability—the core idea of “gravitational mass.”

3. Gate peer-pull with utilization & schedule it: Peer-pull can over-cluster early. Only apply when both experts are above a small EMA usage floor, and ramp β in after warmup.

4. Decay: percentile is good—make it density-aware: Your percentile-based threshold is nice. Add a small repulsion to spread nearby low-usage keys (blue-noise-ish coverage). A cheap way: for each low-usage key, nudge it opposite the local mean of k-NN keys.

5. Capacity & load balancing: Even without token dropping, runtime can balloon if many tokens pile onto one expert. Add a soft capacity penalty directly to scores (no aux loss needed).

6. ANN routing shim for scale: Matmul to all experts is fine for 128–1k experts; it collapses at 10k+ experts. So put the key store behind an ANN index (HNSW/FAISS IVFPQ). Keys move, like so: (a) rebuild periodically, or (b) maintain a small “delta index” for newly moved keys and query both. (Matches the lookup-not-classification argument.

7. Metrics that prove “self-organization,” not just loss beyond Gini/entropy:

- Add a Lorenz curve of expert usage (per layer) and track Gini/Entropy vs. epoch for all layers (not just the middle).
- Neighborhood purity: MI between expert IDs and token facets (language/script, POS buckets).
- Composition score: stability of co-activation cliques across datasets.
- Drift monitors: cosine movement per expert per N steps (flag outliers).

These map tightly to the “galaxies” story in the doc.