Heat on Sym²(X) and LLM Temperature

2026-04-09

Heat on Sym²(X) and LLM Temperature

Our Manifold

`buildManifold` takes N seed points drawn by a user — a curve, a scrawl, a signal — and constructs a surface from all unordered pairs of those points. For each pair (p_1, p_2):

```
vertex = (
  x: (p1.x + p2.x) / 2,   — midpoint x
  y: (p1.y + p2.y) / 2,   — midpoint y
  z: distance(p1, p2)      — separation
)
```

This is Sym²(X): the symmetric product of X with itself. Each vertex encodes a *pairing* — where two points meet, and how far apart they are. High z = far apart. Low z = nearly coincident. Pairs that collapse to the same point land at z ≈ 0, the diagonal.

An adjacency graph connects vertices that share a triangle edge. Neighbors in the graph = pairings that differ by swapping one point to an adjacent position on X.

Heat Diffusion — the Two-Manifold Demo

`two-manifolds.js` runs two heat simulations on the same underlying geometry, side by side.

Diffuse

Both sides share one diffusion operator:

```
function diffuse(h, adj, decay) {
    const next = new Float32Array(h.length);
    for (let i = 0; i < h.length; i++) {
        const nb = adj[i] || [];
        let sum = h[i];
        for (let j = 0; j < nb.length; j++) sum += h[nb[j]];
        next[i] = (sum / (nb.length + 1)) * decay;
    }
    return next;
}
```

Each vertex averages its heat with all its neighbors, then decays by a factor. This is discrete heat diffusion — the graph Laplacian applied once per frame, with exponential decay.

Left — MOAD unpatched (O(n²))

```
for (let k = 0; k < 20; k++) {
    const i = Math.floor(Math.random() * nVerts);
    heat1[i] = Math.min(heat1[i] + 0.5, 8.0);
}
heat1 = diffuse(heat1, adj1, 0.975);
```

- 20 random injection events per frame
- Decay: 0.975 — slow. Heat barely leaves.
- Cap: 8.0 — but with 20 simultaneous hits, many vertices approach cap quickly.
- Displacement: `z += heat1[i] * 120` — amplified hard.

Result: injection rate dominates decay. Heat accumulates. Our manifold warps — vertices spike, the surface becomes chaotic. This is MOAD-0001 at a physical layer: an $O(n^2)$ injection rate that overwhelms a linear drain.

Right — patched (O(1) / O(n))

```
heat2[Math.floor(Math.random() * nVerts)] = Math.min(
    heat2[Math.floor(Math.random() * nVerts)] + 0.4, 1.2
);
heat2 = diffuse(heat2, adj1, 0.90);
```

- 1 injection event per frame
- Decay: 0.90 — fast. Heat dissipates readily.
- Cap: 1.2 — much lower ceiling.
- Displacement: `z += heat2[i] * 40` — moderate.

Result: injection rate and decay reach equilibrium. Heat stays bounded and evenly distributed. Our manifold breathes gently but holds its shape.

What Heat Means on $Sym^2(X)$

A vertex encodes a pairing (p_1, p_2). Heat at that vertex = *unresolved tension between those two points*. High heat = the pairing is energized, stretched, pulling apart. Low heat = the pairing has settled.

When heat diffuses to a neighbor, it spreads tension to adjacent pairings — ones that share a point with the original pair. Tension propagates through our adjacency graph the way uncertainty propagates through a network.

The diagonal ($p_1 = p_2$, $z = 0$) represents collapsed pairings — certainty. Moving away from the diagonal along z = moving into uncertainty, multiplicity, unresolved separation.

LLM Temperature — Same Mechanism, Different Substrate

An LLM produces a logit vector over its vocabulary. Softmax converts logits to probabilities:

```
p(token) = exp(logit / T) / Σ exp(logit_j / T)
```

T is temperature.

- **T → 0**: logits dominate. One token's probability → 1. All others → 0. Our distribution collapses to a point — like a manifold vertex falling to the diagonal (z → 0).
- **T → ∞**: all logits equalized. Every token equally likely. Our distribution flattens — maximum entropy, maximum wobble, no vertex settles.
- **T = 1**: default. Logits used as-is.

The Correspondence

$Sym^2(X)$	LLM temperature
vertex on manifold	token probability
heat at vertex	uncertainty of that token
diffusion across adjacency	correlation between token probabilities
injection rate	how often high-entropy paths are opened
decay	how quickly the model collapses back to a decision
diagonal (z = 0)	argmax — single token, no uncertainty
high-z vertex	near-tie between two candidates
MOAD unpatched manifold	T >> 1, runaway heat, chaotic outputs
patched manifold	T ≈ 0.7–1.0, bounded heat, coherent outputs

Runaway Heat = High Temperature

Our left manifold (MOAD unpatched) does 20 injections per frame with slow decay. The surface warps. This is exactly what T >> 1 does: it injects energy into every logit comparison simultaneously, preventing any pair from settling. Our model never collapses to a decision — it wobbles.

Our right manifold (patched) does 1 injection with fast decay. Heat dissipates before it compounds. This is T < 1: energy drains fast, our distribution peaks cleanly.

Equilibrium Temperature

The patched manifold finds an *equilibrium heat distribution* — not zero (that would be T → 0, argmax, dead manifold), but a steady bounded warmth that lets vertices move without flying apart. This corresponds to the "right" temperature for a task: enough randomness to explore, enough decay to decide.

Finding that equilibrium is the art of temperature tuning. Our manifold shows you what it looks like when you get it wrong in both directions — too much heat (chaos), too little (flatness).

Wobble as a Third Axis

Our kernel's animate loop adds one more layer:

```
        positions[i] = originalVertices[i] + (Math.random() - 0.5) *
(wobbleBase * oracleMod);
```

`wobbleBase` is derived from our input signal's character entropy. High-entropy input → higher wobble. This is stochastic noise applied uniformly — like top-p / top-k sampling: a floor on randomness that operates independently of heat.

Three temperature-like quantities run simultaneously in our kernel:

1. **Heat** — local, diffuses via adjacency, represents relational uncertainty
2. **Wobble** — global, uniform, represents input entropy
3. **Flicker** — sinusoidal opacity, represents attention oscillation

LLMs have analogous layers: temperature (heat), sampling strategy (wobble), and attention dropout (flicker). All three contribute to whether a generation collapses or stays alive.

Do LLMs Use a Similar Algorithm?

Yes — closely. The mechanism is structurally the same. The key difference is where the graph comes from.

Our diffuse:

```
        next[i] = (h[i] + Σ_j h[neighbor_j]) / (neighbors + 1) * decay
```

Weighted average over a fixed adjacency graph. Decay prevents runaway.

Transformer self-attention:

```
output[i] = Σ_j softmax(Q_i · K_j / √d) * V_j
```

Weighted average over a *dynamic* adjacency graph. Same shape, different graph.

our manifold	transformer
fixed adjacency (triangulation)	dynamic attention weights (computed each forward pass)
uniform neighbor average	learned/computed weighting via Q·K
decay factor	layer normalization
heat value at vertex	activation magnitude in residual stream
heat diffuses to neighbors	information propagates token-to-token across layers
injection event	high-signal input token (rare, surprising, semantically loaded)

Where it diverges: our manifold injects heat at random vertices. Transformers inject "heat" from the input itself — a surprising token (low prior probability, high information content) creates a large signal that then diffuses to neighboring tokens through attention in subsequent layers. Early layers carry local heat (syntax, proximity). Later layers carry global heat (semantics, long-range dependencies). Same diffusion shape, but the graph rewires at every token and every layer.

Temperature is only explicitly applied at the *output* layer — it does not govern attention's internal averaging. The √d scaling in Q·K is the attention analog of our decay factor, preventing attention logits from growing so large that softmax collapses to one-hot (which would be $T \to 0$, no diffusion, just routing).

So: yes, LLMs diffuse heat. Our manifold makes the mechanism visible with a fixed graph and explicit parameters. A transformer does it with a learned dynamic graph and calls it attention.

Conclusion

Heat on $\mathrm{Sym}^2(X)$ is not decorative. It is a physical model of uncertainty propagating through a space of pairings. Our manifold is a probability simplex made visible — each vertex a candidate, its heat its weight, its neighbors its correlations.

LLM temperature is heat injection rate vs. decay rate. Our two-manifold demo is that equation, made spatial. Watch our left side warp and know: somewhere a language model is doing the same thing to every token it considers, and calling it creativity.