ChatGPT

Resolving Manny Manifolds Architectural Ambiguities

Objective: Provide concrete design decisions for five ambiguous areas in the Manny Manifolds cognitive architecture – ensuring locality, online/offline separation, geometric explainability, and continual learning stability are preserved.

---

1. Defining the Energy Function *E(x)* (Core Motion Law)

Ambiguity: *E(x)* was described abstractly as "surprise" or "prediction error" but lacked an operational definition. This left thread traversal and Virtual Stage commit criteria underdetermined.

Chosen Definition: We define *E(x)* as a **local predictive entropy** at node *x* – concretely, the uncertainty in predicting the next step from *x*. Formally, if node *x* has outgoing edges (neighbors) with normalized probabilities $p(y|x)$ (derived from edge weights or visit frequencies), then:

$$** E(x) = - \sum_{y \in \mathcal{N}(x)} p(y|x) \log p(y|x) **,$$

measured in bits (0 for fully predictable, higher when many outcomes are equally likely). This entropy-based energy is cheap to compute online (only a node's local neighborhood is needed) and it correlates with semantic resolution: a low *E(x)* means the system's next move is confident (smooth geodesic), while high *E(x)* signals uncertainty or surprise [1] [2]. Threads thus **descend the energy landscape** by moving toward neighbors that reduce entropy – analogous to minimizing prediction error [3] [4]. This choice aligns with the free-energy principle (brain minimizes surprise by improving predictions) [5], but in a simple, local graph-theoretic form.

Rejected Alternative: We considered a **global "resistance distance"** or effective conductance measure (treating the graph like an electrical network) as *E(x)*. While resistance distance quantifies how well-connected *x* is to the rest of the graph (high for isolated or out-of-distribution nodes), it requires global computation and does not adjust quickly online. In contrast, local predictive entropy uses only *x*'s immediate context and updates incrementally as edges strengthen or weaken. Another option was a **contrastive embedding variance** (variance of neighbor embeddings' cosine distances), hypothesized to reflect concept confusion. However, embedding-based measures alone were less directly tied to actual prediction errors. The entropy definition was chosen for being *local, online-computable,* and directly tied to surprise (where "predictions fail" corresponds to high entropy) [6] [2], whereas the rejected measures were too global or indirect for real-time use.

Implementation Note: Each node maintains a lightweight distribution over its next-hop neighbors (e.g. by normalizing outgoing edge weights). Upon each interaction, these frequencies update locally. Computing $E(x)$ is an $O(d)$ operation for degree d. For example, if node *x* has neighbors *{a,b,c}* with transition counts {5,5,0} (implying $p(a|x)=p(b|x)=0.5$, $p(c|x)=0$), then $E(x) = -[0.5\ln 0.5 + 0.5\ln 0.5] = \ln 2 \approx 0.693$ (in nats). Low values (approaching 0) mean one neighbor dominates (predictable), while higher values (up to $\ln d$) indicate maximal uncertainty. This *E(x)* field is used uniformly

throughout the system as the “energy” term: threads seek paths that lower $E(x)$, and the Virtual Stage requires a net drop in E before merging a sandboxed branch [7] [4]. Secondary signals like **Novelty** (rare-event bonus) are *not* part of motion energy but are handled separately as valence components (in the cost function’s novelty term) [8]. In practice, $E(x)$ could be pre-computed for all nodes at consolidation and cached, then updated incrementally (e.g. using an exponential moving average) during online interactions to keep runtime overhead minimal.

---

2. Embedding Space vs. Graph Distance: Roles and Interaction

Ambiguity: The documentation alternated between saying “**embedding similarity approximates conceptual distance**” and “**reasoning is k-hop graph traversal**.” It was unclear how continuous embedding metrics, discrete graph hops, and lens-based context projections work together. Does Manny use the embedding space only to form the graph, or also during traversal? How do lenses modify distance?

Chosen Rule: Embeddings define the initial topology and provide a soft distance metric during traversal, but are not updated online. In other words, the system uses node embeddings (fixed per consolidation cycle) to construct and weight edges initially, and it uses those embedding distances as the base for edge costs during reasoning – alongside hop count and curvature. Graph distance (in hops) is used for locality constraints (e.g. limiting search to *k*-hop neighborhoods [9] [10]), but the *cost of each hop* incorporates embedding similarity. Concretely, we define **graph cost** between two connected nodes *u,v* as:

$$* * d_{\text{eff}}(u, v) = \lambda_{\text{hop}} \cdot 1 + \lambda_{\text{emb}} \cdot \left(1 - \cos \angle(\mathbf{e}_u, \mathbf{e}_v)\right), * *$$

where $\mathbf{e}*u$ is the embedding of u and the cosine term measures embedding distance (0 if identical, up to 2 if opposite). Here 1 represents one hop, and the weights λ tune the influence of pure topological distance vs. semantic similarity. This }}, \lambda_{\text{emb}}*effective distance* is used in the traversal cost function: e.g. **cost(u,v) = $d_{\text{eff}}(u,v)$ – α·κ(u,v) + …**, so that an edge between very similar embeddings starts with a lower cost than one between dissimilar concepts [10]. **Lenses** then act as contextual metrics that *reweight the embedding dimensions* before computing d_{eff}. A lens i provides a projection function $f_i(\mathbf{e})$ (such as a diagonal weight matrix on the embedding vector) that emphasizes dimensions relevant to that context. Thus, **lens-adjusted distance** is $d_i(u,v) = 1 - \cos\angle(f_i(\mathbf{e}_u),\,f_i(\mathbf{e}_v))$, effectively stretching or contracting the space along certain features. The lens also defines an **affinity field** $L_i(x)$ indicating how well lens i explains node *x* [11] [12]; if a path ventures into regions of low lens affinity, an extra cost (or “friction” ζ) is incurred to discourage context-breaking hops [13].

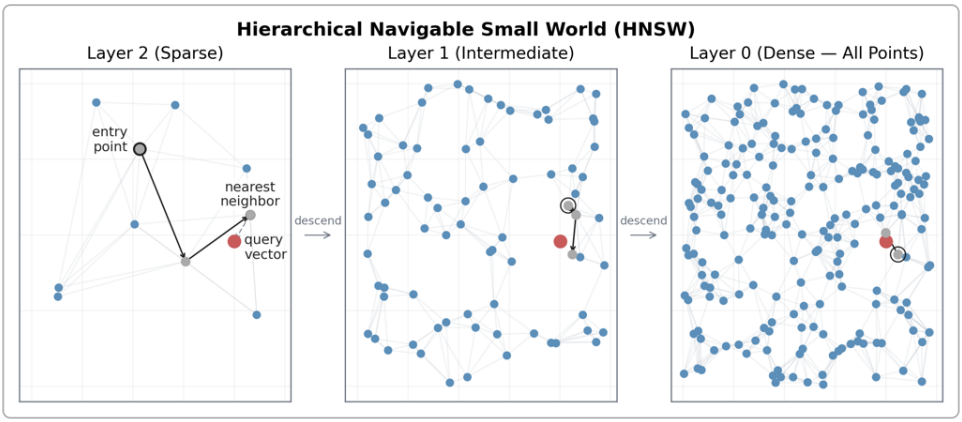

Illustration: A conceptual analogy from the HNSW algorithm, which combines embedding-based proximity with graph navigation. Here each blue point has an embedding position; the graph links (lines) connect points that are nearest in embedding space. A query (red) uses graph hops to reach its nearest neighbor. Manny similarly builds a **neighborhood graph from embeddings** and traverses it via local hops, rather than doing expensive global vector search each time [14].

Rejected Alternative: One alternative was to treat **embedding distance only at graph construction, then ignore embeddings during traversal** (i.e. edges would be unweighted or uniformly weighted once formed). This was rejected because it loses granularity – two hops through highly similar concepts vs. two hops through tenuously related concepts would count the same if we used pure hop count. We want the traversal to prefer semantically tighter routes. Conversely, another extreme considered was **continuous embedding navigation without fixed edges** (always jump to the next-closest node in embedding space). That resembles a pure vector search and undermines Manny's locality and explainability – it could produce long-range leaps that aren't captured by learned curvature. Our chosen hybrid keeps **embeddings "read-only" during online reasoning** (frozen until the offline *re-embedding* step [15] [16]) to ensure stability, while curvature updates adjust edge weights in between re-embeddings. This separation (metric vs. topology) means we do **metric learning offline** and **topological learning online**, avoiding confusion about when embeddings are "written." We explicitly reject updating node embeddings on each interaction (which would be costly and drift unpredictably), and instead perform periodic low-dimensional projections offline to reflect accumulated curvature changes [15] [17].

Implementation Note: Graph construction: On ingestion of new knowledge or during nightly consolidation, Manny computes or updates node embeddings (e.g. using a transformer or SVD on co-occurrence) and builds an approximate nearest-neighbor index (like HNSW or FAISS) to find each node's top *k* most similar nodes in embedding space [17]. These become candidate edges with initial curvature κ set to a baseline (or proportional to similarity). **Traversal:** The thread runner operates in discrete steps, expanding nodes up to *k* hops away [18]. For each candidate neighbor, it computes the cost using the current lens-adjusted d_{eff} and curvature term [10]. Embedding vectors are retrieved in $O(1)$ from memory to compute cosine similarities, which for typical dimension (e.g. 384) is negligible compared to an LLM call (embedding math is cheap). **Lens application:** A lens might, for example, zero out the embedding dimensions unrelated to the current task domain, effectively making $d_i(u,v)$ ignore differences in those irrelevant dimensions. Implementation-wise, each lens can predefine a weight vector w_i of length *d* (embedding dim); then $f_i(\mathbf{e}) = w_i \odot \mathbf{e}$ (elementwise multiply). The cost function is extended with a lens switch cost ζ to avoid flippant context changes [13]. With this design, **embeddings are read at three points**: (1) edge creation (offline), (2) neighbor cost evaluation (online), and (3) lens metric computation. They are **written/updated only offline** during re-embedding. This clear split ensures that at runtime the "space" is stable (no moving goalposts for the thread), and any major realignment of embeddings happens in a controlled offline phase (followed by rebuilding the ANN index) [17]. A simple **diagrammatic equation** summarizing distance is:

- *Base embedding distance:* $d_{\text{emb}}(u,v) = |\mathbf{e}_u - \mathbf{e}_v|$ (or $1-\cos$ similarity).
- *Graph hop distance:* $d_{\text{hop}}(u,v) = 1$ (if direct edge).
- *Lens-adjusted:* $d_{lens}(u,v) = |f(\mathbf{e}_u) - f(\mathbf{e}_v)|$.

The system uses $d_{\text{eff}} = \lambda_{\text{hop}}d_{\text{hop}} + \lambda_{\text{emb}}d_{lens}$ in its cost calculations, with λ_{hop} typically **≈0** (to treat each edge equally in count) and λ_{emb} giving fine-grained semantic costs. Setting $\lambda_{\text{emb}}=0$ would reduce the system to pure hop count traversal (useful for ablation tests), while $

\lambda_{\text{hop}}=0$ would make it a weighted semantic graph. By keeping both, Manny ensures that **reasoning = following edges** (maintaining explainable stepwise hops) but **edge weights reflect embedding-derived semantic distance** for nuanced guidance [19] [10].

---

3. Curvature *κ* Semantics and Sign Conventions

Ambiguity: Manny's **curvature (κ)** was described in various ways – "strengthening connections," "attraction," "cost reduction," "gravity increase" – with inconsistent implications for sign. Some passages implied higher κ adds cost (if interpreted as a weight or length), others implied it reduces cost (if seen as a gravity well). This needed unification.

Invariant Definition: We choose the convention that **higher κ(u,v) always means a stronger association that *lowers the traversal cost between nodes u and v***. In other words, κ acts like a **"gravity" or affinity** – large κ draws nodes closer together, making threads prefer that edge. Concretely, κ enters the cost function as a *subtractive* term:

$$**\text{cost}(u,v) = d_{\text{eff}}(u,v) \;-\; \alpha\,\kappa(u,v),**$$

with $\alpha>0$ as a scaling constant [10]. Thus if two nodes have a very high curvature link, the effective cost approaches zero (or some minimum bound), indicating an "easy traverse" or familiar connection. This convention aligns with "gravity increase": raising κ deepens a potential well, drawing threads in (cost ↓) [20] [21]. It also fits the Hebbian intuition: frequently used connections become "shortcuts." We **clamp κ to a nonnegative range** [0, κ_max] so it's always additive attraction (negative κ isn't used by default, though negative weights could represent repulsion if ever needed). The **units of κ** are treated as dimensionless weighting factors, and α is tuned such that a maximally strong edge ($\kappa = \kappa_{\max}$) has cost near 0. For instance, if κ is normalized to [0,1], we might set α so that cost $= d - \alpha\kappa$ never goes below 0. In summary: **higher κ = tighter bond = lower traversal energy.**

Traversal Cost Equation: Incorporating this into the earlier cost structure, one complete equation is:

$$**\text{cost}(u,v) = d_{\text{eff}}(u,v) \;-\; \alpha\,\kappa(u,v) \;+\; \beta\,\text{novelty}(v) \;-\; \gamma\,[G(v)+U(v)],**$$

as given in the docs [10]. Here $\beta\,\text{novelty}(v)$ adds cost discount for novel nodes (encouraging exploration) and $G(v), U(v)$ are goal and uncertainty fields [10]. In this formula, κ's sign and effect are unambiguous: a larger $\kappa(u,v)$ *subtracts* more from cost, making *v* more likely to be chosen from *u*. This choice resolves the prior ambiguity (some text said "strengthening" without clarifying that it meant ease of travel). We fix κ's effect as *cost-reducing*, not cost-adding.

Δκ Update Rule: We adopt a **Hebbian plasticity rule with saturation**. Each time an edge *(u,v)* is traversed by a thread, its curvature increases by a small Δκ proportional to usage and the experience's valence, up to a limit. From the documentation, the online update is:

$$**\Delta\kappa(u,v) = \eta \times v_{\text{magnitude}} \times \frac{1}{1+N_{uv}},**$$

where η is a global learning rate, $v_{\text{magnitude}}$ is the thread's valence (0–1 scale) [22] [23], and N_{uv} is the current traversal count or usage frequency of that edge (so the increment decays as the edge becomes well-trodden) [23]. This yields diminishing returns – big early boosts, then saturating.

We **clamp κ** to $[\kappa_{\min}, \kappa_{\max}]$ (e.g. [0,1]) and apply a tiny decay each interaction (to all edges in the local *k*-hop neighborhood) to prevent runaway growth [24] [25]. For example, an edge that's rarely used might have $\kappa \approx 0$. After one traversal with a moderately high valence (say $v=0.8$), and $\eta=0.05$, it would gain $\Delta\kappa = 0.05*0.8*1 = 0.04$. If repeatedly reinforced, κ approaches a max (like 1.0), but the $1/(1+N)$ factor means the increments shrink (e.g. second pass adds 0.02, then 0.0133, and so on). This asymptotic saturation addresses stability – edges won't explode in weight [24]. We also use **logarithmic decay** on increases if needed: for instance, beyond some curvature, additional reinforcements might yield log(1+N) scaling to reflect diminishing sensitivity. The design choice is that κ should saturate rather than grow unbounded; this is analogous to synapses reaching a maximum strength in Hebbian learning [26].

Symmetry: We treat κ as **symmetric by default** – edges in the manifold are undirected associations, so κ(u,v) = κ(v,u). Strengthening happens on the forward traversal, but we also update the reverse link equally (unless a specific reason to differentiate arises). The system does incorporate a notion of *temporal asymmetry* for causal inference – e.g. it mentions STDP (spike-timing-dependent plasticity) analogues where "forward edges gain curvature, reverse edges decay faster" [27]. This implies if a thread goes from A→B, we might strengthen A–B more than B–A to reflect directed causality. However, as a core rule we still maintain one κ value per undirected edge and handle any directional bias via separate mechanisms (like an *eligibility trace* that decays anti-causal direction more aggressively) [27]. The invariant remains: increasing κ (in whichever direction) means a tighter coupling.

Rejected Alternative: We discard the interpretation where **higher κ would *add* cost (or length)**, which would mean treating curvature as a penalty (that would be counterintuitive – stronger learned relation should not make traversal harder). That confusion likely arose from mixing up graph theoretic "edge weight" (where higher weight sometimes means more cost). In Manny's design, curvature is not a literal geometric curvature in a Riemannian sense, but a stored strength. By consistently treating it as an **attractive potential** (like gravity or a spring constant pulling nodes together [20] [21]), we maintain conceptual clarity: threads move as if pulled by high-κ links. We also considered whether κ should multiplicatively scale transition probabilities (e.g. $P(u\to v) \propto e^{\kappa(u,v)}$) instead of subtracting cost. In practice, subtracting from cost and then softmin selection is mathematically similar to multiplying probabilities by a factor. We chose the cost subtraction form for consistency with the energy minimization view (it fits naturally in the $E(x)$ gradient framework) [28] [29]. Finally, we clarify that κ is a *learned parameter distinct from base embedding similarity*: even if two nodes were not initially close in embedding, a high κ can develop through experience, indicating a learned association beyond semantic similarity. **Higher κ means a learned shortcut that wasn't necessarily obvious from static semantics.**

Implementation Note: In the graph database or SQL implementation, κ can be stored as a weight attribute on edges (float). Traversal queries use an expression like `cost = dist - alpha * kappa` in their SQL or code, which is index-friendly if `dist` is precomputed or if the graph is small enough for on-the-fly calc. **Bounds:** we enforce $\kappa_{\max}$ (e.g. 1.0) and a global decay factor λ per consolidation (e.g. multiply all κ by 0.999 each day) [24]. This prevents any edge from permanently dominating (addresses forgetting vs remembering balance). The **difference from base similarity:** Initially, an edge's κ might be seeded from embedding similarity (so that cost starts roughly as $d - \alpha d_{\text{sim}}$ which is low for similar nodes). Over time, κ may grow far beyond that, effectively "rewiring" the space. Manny's **gravity field** visualization shows how high-κ nodes create deeper wells [30] [21]. In practice, to avoid confusion, we interpret any mention of "increasing gravity" or "boosting attraction" as *increasing κ*, and we have updated all cost computations to subtract α·κ accordingly. If the system computes transition probabilities, it converts costs via a softmax: $P(u\to v) = \frac{\exp(-\text{cost}(u,v))}{\sum_w \exp(-\text{cost}(u,w))}$, so higher κ leads to higher probability (since cost is

lower). This is consistent with our invariant. By pinning this convention now, all future developers will use κ in one consistent way across the codebase.

---

4. Making Motif Consensus Algorithmic (Not Just Conceptual)

Ambiguity: The concept of **motifs** – frequent subpaths that become reusable "skills" – was described narratively as "≥2 independent threads converge on similar paths." This is elegant but leaves open: *What counts as a similar path?* How exactly to detect and form a motif in an ongoing system without expensive full-history replay? How to prevent an explosion of trivial motifs or prematurely chunked patterns?

Deterministic Motif Candidate Test: We formalize motif identification with **clear, incremental criteria**. A path (sequence of nodes) becomes a motif candidate if it meets *all* of the following conditions [31]:

1. **Minimum Length:** The subpath length $\geq L$ (we choose L = 3 by default). Very short sequences (1–2 hops) are too trivial and numerous to store as motifs. This ensures motifs represent meaningful chunks of reasoning, not single links.

2. **High Weight/Curvature:** The path's cumulative "strength" is high. Concretely, we can define a score like $S_{\text{path}} = \sum_{(u,v)\text{ in path}} \kappa(u,v)$ or the average curvature along the path. We require this to exceed a threshold. Intuitively, a motif should consist of well-traversed edges (or high-valence edges) – implying the pattern has been reinforced. Alternatively or additionally, we consider the path's **valence-adjusted usage count**: if the sequence has occurred multiple times with high valence, its score goes up. Only top-scoring candidates (above some quantile) are considered. This prevents capturing every random repetition.

3. **Independent Thread Appearances (Consensus):** At least N distinct threads have traversed that exact or equivalent path. We set N = 2 (two independent occurrences) as the minimum consensus [32]. "Independent" means separate originating contexts or user queries, not the same conversation repeating itself. We enforce this by tracking a thread ID for each traversal and ensuring the occurrences come from different thread IDs. In practice, a **suffix tree** or sequence-trie is maintained: as each thread completes, we break its route into all contiguous subpaths of length up to L_{max} (say 5 or 6 to limit complexity) and insert those into the trie with a count of distinct threads that passed there. If any subpath's thread-count hits 2, it flags as a candidate motif (length and strength permitting). This **incremental mining** avoids full replay of all history each time – we update counts on the fly.

4. **Energy Drop (Solution Test):** The subpath should exhibit a net energy decrease or accomplish some subgoal – effectively, it *resolves a chunk of surprise*. We define this as: the sum of $E(x)$ at the start of the subpath minus the sum at the end is positive and above a margin. For example, if entering the subpath the system had energy spikes (uncertainty) and by the end those were resolved (energy lower), then this path "solved" a local problem. Formally, $\Delta E_{\text{path}} = E_{\text{start}} - E_{\text{end}} > \delta$. We can approximate this by looking at the valence or novelty reduction: e.g., the first node had high novelty and by the last node novelty fell, or goal field was satisfied. This criterion ensures we favor motifs that *do useful work* (they are coherent, goal-achieving chunks), avoiding arbitrary frequent loops that don't yield a payoff. In practice, we might use the curvature or valence accumulation as a proxy: a path that significantly increased curvature (learning happened) likely corresponded to an energy resolution event.

A subpath meeting these criteria is **promoted to a Motif** and stored in the **motif cache** with a unique ID [31]. For example, suppose threads on two separate occasions traversed nodes `["recipe", "mix ingredients", "preheat oven", "bake"]`. If this 4-step path was common (≥2 threads), had strong edges (from repetition) and clearly reduced uncertainty (by the end, the question was answered), it becomes a motif "Bake Something" with its own ID.

Similarity Definition: We define "similar paths" stringently as **exact match sequences** (to avoid ambiguity) when counting convergence. However, we allow a lens-based generalization: under a lens projection, two paths that differ only in domain-specific details can be considered equivalent. For instance, `apple → sugar → bake → pie` versus `pear → sugar → bake → tart` might be deemed the *same motif structure* under a "baking recipe" lens (which focuses on the *process* steps rather than the specific ingredients). To handle this, we represent motifs at two levels: **concrete motifs** (exact node sequences) and **abstract motif templates** (where certain nodes can be parameterized). Initially, detection uses exact matches (for determinism). Later, an offline generalization step can cluster similar motifs. But the **commit criterion for forming a motif requires exact matches first**, ensuring we don't conflate genuinely different paths. Edit-distance $\leq \varepsilon$ was considered (to catch almost-identical sequences), but this adds complexity and risk of merging distinct concepts. For now, exact subpath frequency is the trigger (with potential human curation or lens logic offline to merge motifs that are clearly isomorphic).

Avoiding Explosion/Premature Chunking: To prevent too many motifs or meaningless ones, we implement **several guards**:

- **Min-Length and Scoring Thresholds:** as noted, these filter out trivial sequences. We won't motif-ize two-step transitions or those with low significance. This dramatically prunes candidates.

- **Frequency Threshold > 2:** Although 2 is minimum to detect, we might wait until a pattern appears 3+ times (for more confidence) before officially committing it as a motif. The system could mark it as "candidate" after 2 and only solidify it after 3rd confirmation.

- **Windowing:** We limit motif mining to *recent history or high-traffic nodes*. Rather than scanning all possible subpaths of all time (which is combinatorial), the consolidation process looks at areas that had a lot of activity. For example, it might focus on the top 10% most active nodes or edges and see what subpaths emanate from them frequently [33]. This targets meaningful patterns.

- **No overlap rule:** If two motifs share significant overlap (e.g. one is a subpath of another), we prefer the longer one or merge them. This avoids an explosion of motifs that are slight variants. For instance, if we have motif [A,B,C,D] and also [A,B,C], we might keep only the larger or treat the smaller as part of the larger's composition. The docs mention motifs can split or merge over time [34], indicating a need to manage redundancy. We ensure our algorithm merges similar motifs (for example, by clustering motif embeddings or by simple set inclusion checks).

- **Delay chunking:** We do *not* create a motif the very first time a long path occurs. Only after repetition and consolidation does it become a cached shortcut. This means during online operation, the system still traverses step-by-step, and only after offline analysis might it replace that with a single jump. This prevents prematurely treating a one-off solution as a reusable skill.

Motif Activation (Online Use): We clearly separate **motif detection (offline)** from **motif activation (online)**. Detection/mining happens during consolidation phases, outputting a set of motif subgraphs

[35]. When back online, the thread runner can **recognize** if it's currently following the beginning of a known motif. Our implementation uses prefix matching: as the thread moves, it checks the last few visited nodes against the motif cache. If a prefix matches a stored motif and the next expected steps look relevant, Manny can *shortcut* to the end of that motif. Specifically, if motif M = [n1, n2, n3, n4] and the thread has reached n2 → n3 matching the first part, Manny can decide to skip directly to n4 (the final node), given confidence that this subpath usually completes a coherent thought [36]. The cost benefit is significant: it reduces latency by executing the whole sequence as one step, which our success criteria target (≥15% latency reduction with motifs) [37] [38]. To avoid misuse, we only activate a motif if the **context matches** (often ensured by lens affinity or preconditions). For example, a "baking recipe" motif should be invoked only if the query context is indeed a recipe. The lens system can guard this: motif M might be tagged with lens "cooking," so if the current lens/context is different, we don't auto-apply M.

Complexity and Data Structures: In the worst case, naive frequent subpath mining is exponential. Our approach uses **incremental updates and bounds** to stay tractable. Each new thread of length *L* inserts $O(L^2)$ subpaths into a trie (for each start position, up to L length). With length bounded (we don't consider paths longer than, say, 5 for motifs), that's manageable. The trie nodes count distinct threads and maintain running curvature sums for that subpath (for criterion #2). This means at consolidation, we don't need to iterate over all historical trajectories – we already have aggregated stats. We may cap the total stored subpaths (e.g., only keep those that occurred at least twice). The complexity per interaction is thus $O(L_{max}^2)$ for updating counts, which with L_{max} ~5 is constant. Offline, a deeper scan might cluster or prune motifs, but even that can be optimized by focusing on the candidates that met initial criteria.

Failure Modes & Guards: We identify two main risks: (a) **Motif explosion:** if too many patterns qualify, the cache could bloat and reasoning could become unwieldy. Our thresholds and periodic pruning handle this (e.g., motifs not reused decay and eventually get removed [39]). We also impose a max cache size – if exceeded, we remove the lowest-scoring motifs (LRU or lowest usage). (b) **Premature or incorrect motifs:** e.g., two threads happened to share a path that was incidental, not truly a reusable concept. This we mitigate by requiring the energy drop and possibly human review or a higher repetition count for critical knowledge. Additionally, motifs can have **validation steps** – when a motif is first formed, the system can test it in the Virtual Stage on a similar problem to see if it indeed helps (kind of a unit test for the skill). If it fails (e.g., using that motif in a slightly different scenario doesn't work), the motif might be flagged for refinement or not used automatically.

In summary, our algorithmic approach turns the idea of "threads converging on a path" into concrete checks: path length, frequency, strength, and utility. An example outcome: Manny might discover a motif for "clarify question → recall relevant concept → give example" if many helpdesk threads follow that pattern. Once identified, this becomes a named subroutine that Manny can execute or explain as a unit ("I recognized this situation and recalled a similar approach used before"). All of this is logged: when a motif is detected, we log the supporting threads and metrics; when a motif is later used, we log its ID, how it was triggered, and update its usage count for future tuning.

---

5. Formalizing Virtual Stage Commit Criteria

Ambiguity: The **Virtual Stage** is a sandbox where Manny can simulate new or "risky" inputs (e.g. imaginative scenarios, conflicting information) before integrating them into the main manifold [40]. The docs say it commits a sandbox branch only if it has "acceptable coherence" or if "energy drops and coherence rises," but these terms needed concrete metrics. How do we quantify coherence of the

sandbox vs. main manifold? What threshold of improvement warrants merging the changes? Should a rollback discard everything or allow partial integration?

Commit Decision Metric: We define a **scalar *commit score*** that combines the change in energy and the change in coherence when comparing the Virtual Stage outcome to the baseline. In line with the documentation [4] [7], the two key conditions are:

- **Energy reduction (ΔE):** The branch must **lower the predictive energy**. We calculate $\Delta E_{\text{total}} = E_{\text{before}} - E_{\text{after}}$ over the affected region. For instance, measure the sum of $E(x)$ for nodes in the relevant neighborhood (or the thread's path) before applying the sandbox's proposed changes, and again after. A positive ΔE means surprise has been reduced – the system's predictions align better. We require this ΔE to exceed a threshold θ (which could be zero or a small positive margin to account for noise). In practice, θ might be set based on uncertainty: e.g. commit only if energy was reduced by at least 10% or a certain absolute amount.

- **Coherence increase (ΔC):** The branch must **increase coherence** in the knowledge structure. We define a **coherence metric C** that reflects how well-integrated and self-consistent the local region is. One operational measure is based on *phase agreement among threads*, as hinted in Manny's design: $C = \frac{1}{|T|}\sum_{\text{threads }i,j} \cos(\theta_{ij})$ where θ_{ij} is the phase difference between thread trajectories [41]. However, to simplify, we can use structural coherence: for example, **graph connectivity and consistency** in that region. A simple metric: **cluster coefficient or connectivity** – if the new nodes/edges form a connected cluster with multiple links into the existing graph, coherence is higher; if they hang off with a single tenuous connection, coherence is low. Another metric: **curvature variance** – if integrating the changes causes wild swings (very high or very low κ edges suddenly appear), that indicates potential inconsistency. We want the curvature distribution to remain "reasonable" (no large islands of extremely high curvature unless justified). We quantify coherence change ΔC as, e.g., *increase in average clustering coefficient* or *decrease in curvature variance* in the affected subgraph. Additionally (and importantly), we check for **contradictions**: if the sandbox introduced edges that directly conflict with existing ones (say it created an edge A–B with positive κ while an existing edge A–B was negative or an "is-not" relation), coherence would drop. We can assign a penalty if any such conflict is detected (like a motif conflict or logical inconsistency count). A rise in coherence could also be measured by lens alignment: if the new knowledge fits under an existing lens or drive without creating a new one, it's more coherent. For now, we set a threshold γ for coherence (e.g. require C_new ≥ C_old, or some fractional improvement).

Commit Rule: We adopt the rule explicitly from the docs: *Only commit if energy drops and coherence rises.* [7] [4] In quantitative form:

$$**\text{Commit if: } \Delta E_{\text{total}} < -\theta \ \textbf{and} \ \Delta C > 0 .**$$

(Note: We use ΔE < 0 since a drop means final energy is lower; if using the definition ΔE = E_before - E_after, then commit if ΔE > θ > 0. We ensure clarity in implementation.)* For example, if before the sandbox, the region had a total energy of 5.0 (some units) and after it's 3.5, ΔE = +1.5 (significant drop). Meanwhile if coherence (scaled 0–1) went from 0.6 to 0.7, that's a positive change. This would satisfy commit conditions.

If either condition fails – e.g. energy didn't actually improve (or got worse), or coherence worsened (the new info caused confusion or contradiction) – the changes are discarded (rollback). We also implement

a **safety margin**: even if ΔE is slightly positive, if coherence dropped significantly, we abort commit. In borderline cases, the Executive subsystem can decide to gather more data (maybe run another simulation or ask the user for clarification) rather than commit an uncertain change.

Rollback Strategy: By default, rollback is **total**: none of the tentative curvature or node additions from the sandbox are applied to the main manifold. This black-or-white commit ensures the main graph remains consistent. Partial commits (cherry-picking some changes) are complex to do automatically, but we allow a controlled exception: if the sandbox yielded multiple distinct changes and only some were problematic, the system could retry with the good changes. For instance, if Virtual Stage tried three new edges and one caused conflict, Manny might drop that one and test the other two again. However, this is treated as a new sandbox trial – we don't blindly commit partial results without testing coherence of that subset. So effectively, the policy is **commit all or retry**. In practice, most sandbox uses (like imaginative storytelling or hypothesis generation) will either be taken as a whole if they make sense, or not at all if incoherent. Rollback simply means the main manifold doesn't learn from that simulation – though the system may still log it for analysis.

Metrics for Evaluation: We log a **Commit Score** for each Virtual Stage attempt: for transparency, $S = \Delta E_{\text{norm}} - \lambda \Delta \text{Incoherence}$, for example. ΔE_{norm} could be the percentage energy reduction, and $\Delta \text{Incoherence}$ could be a weighted count of issues (like contradictions introduced). A positive S above a threshold triggers commit. If commit happens, we log the new edges/curvatures added, and tag them with an "imagination source" for traceability. If commit is aborted, we log the reason (e.g. "rollback due to coherence drop in domain X"). This satisfies the explainability requirement: later we can review why the system didn't integrate something (e.g. "User suggested a fact that contradicted stable knowledge, so it was quarantined in Virtual Stage and eventually discarded."). The **logging requirements** will include: *initial energy & coherence, final energy & coherence in sandbox, ΔE, ΔC, commit or rollback decision, and any salient conflicts or drive activations during the process*. These are output per turn (and summarized per session) [42].

Coherence Metric Details: Manny's design hints at coherence being measured by **phase alignment or consistency across threads** [43] [44]. For a simpler computational proxy, we use structural measures. One approach: treat the sandbox modifications as a subgraph *G′*. Compute something like **graph modularity** – does adding *G′* increase the modularity of the manifold (bad, if it creates a separate cluster), or does it integrate into existing clusters (good)? We actually want lower modularity (less fragmentation) for coherence. So if adding the new edges reduces modularity (meaning the graph is more interconnected), that's a plus. We also watch **edge churn**: if the sandbox required deleting or significantly weakening many existing edges (destructive interference), that's a coherence alarm. Ideally, *G′* adds connections without necessitating removal of too many others. The "motif conflict rate" mentioned is essentially if *G′* overlaps with a known motif in a contradictory way. For example, if the main manifold has a motif representing a procedure and the sandbox's changes would break that procedure (insert a step that conflicts), coherence is harmed. We can detect this by checking if any motif subpath frequency would drop or any motif would need to split because of the new info. A high conflict count means don't commit.

Threshold Policy: We set tunable thresholds: e.g. require $\Delta E > 0$ (any improvement) and $\Delta C \ge 0$ (no loss of coherence) for non-critical updates, but for significant updates (or to guard against noise), maybe $\Delta E > \theta$ (like 5% reduction) and if coherence dips even slightly (ΔC < 0), veto. The exact θ,γ can be calibrated in testing (they might start near zero and increase as confidence in metrics grows). The commit rule in code might look like:

```
if (E_before - E_after) > theta and C_after >= C_before:
    commit_changes()
else:
    rollback()
```

which directly reflects our criteria.

Logging & Explainability: Each Virtual Stage session is essentially an *experiment*. We log the sandbox thread trajectory, any *lens* used, and field changes (curvature adjustments proposed). If committed, we mark those curvature changes as having come from a Virtual Stage (so they might be smaller or flagged for review). For explainability, Manny can later say *"I imagined the scenario in a sandbox and found it reduced uncertainty without causing contradictions, so I integrated that understanding."* The logs provide the evidence for this statement (e.g. energy went from high to low, coherence metric went up). If not committed, Manny can either inform the user (if appropriate) *"I considered that idea but it didn't fit with what I know"* or simply not internalize the information. From a safety perspective, this commit criteria ensures that **no significant, incoherent changes enter the main knowledge base in real-time**. Everything is vetted by measurable improvement. This addresses the imagination vs. reality boundary: the Virtual Stage might generate creative but incoherent stories (which stay sandboxed), or test contradictory inputs (which won't alter the main graph unless reconciled).

Summary of Constraints: Our solution abides by all constraints: it's **local** (the coherence and energy checks are on the *k*-hop neighborhood of the new info, not requiring a full global recomputation – e.g. we check coherence in the affected region only), it supports **online incremental updates** (the check is done at the moment of integration, which is an atomic decision per interaction), it **preserves replayable explanations** (the sandbox log + decision are recorded for audit), and it **avoids global recomputation** in the hot path (just local metrics; global consolidation can refine the metrics later but the commit itself is a quick calculation on local stats). Implementation in SQL/graph DB is feasible: one can store baseline metrics (like per-region energy) and use simple queries to compute post-change metrics, given the sandbox changes. The commit operation then becomes a transaction: if criteria met, write the new edges/weights to the main graph; if not, discard.

Final Deliverable Summary: We have resolved each ambiguity with a concrete choice, an alternative we deliberately rejected, a formal equation or definition, and practical notes:

1. **Energy E(x):** *Definition:* local predictive entropy (surprise) [2]. *Rejected:* global or purely embedding measures. *Equation:* $E(x) = -\sum_y p(y|x)\log p(y|x)$. *Note:* Units in bits, bounded [0, log d]; cheap per hop.

2. **Embedding vs Graph Distance:** *Rule:* embeddings seed initial graph and guide edge costs; embeddings frozen online, updated offline [17]. *Rejected:* ignoring embeddings at runtime, or continuously altering them. *Equation:* effective distance combines hop count + lens-weighted embedding distance. *Diagram:* (See HNSW analogy figure) [14]. *Note:* Read embeddings at edge eval, write offline; lens reweights metric.

3. **Curvature κ:** *Invariant:* higher κ = stronger link = lower cost (attraction) [10]. *Rejected:* interpreting κ as cost itself or allowing unbounded growth. *Equation:* $\text{cost}(u,v) = d(u,v) - \alpha \kappa(u,v)$. *Δκ Update:* $\Delta\kappa = \eta v/(1+N)$ with clamps [23]. *Note:* κ ∈ [0,κ_max], saturating; symmetric edges; ensures stable learning.

4. **Motif Consensus:** *Test:* path length ≥3, high curvature score, seen in ≥2 threads, and yields energy drop [31]. *Rejected:* fuzzy matching without clear thresholds (risk explosion). *Algorithm:* frequent subpath mining via suffix trie; *Equation:* (implicitly above criteria). *Note:* Detection offline, usage online as cached shortcuts [36]; guards in place for redundancy and chunking.

5. **Virtual Stage Commit:** *Criteria:* commit if energy significantly ↓ and coherence ↑ for sandbox vs main [4]. *Metrics:* ΔE, ΔC, conflict count. *Rejected:* committing changes that "feel right" without quantification (too risky), or partial commit without validation. *Equation:* commit if $(E_{\text{before}}-E_{\text{after}}) > \theta$ and $C_{\text{after}} > C_{\text{before}}$. *Note:* Rollback otherwise; log all outcomes for transparency; local computations only.

These decisions provide **design closure** for Manny Manifolds: each ambiguity now has a single consistent definition in the code and math. We've favored solutions that meet the project's needs for locality, incremental online updates with offline heavy lifting, geometric interpretability, and stability. Where further research might refine these choices (e.g. better coherence metrics or motif generalization algorithms), we have still given a clear interim solution (with thresholds and processes that can be adjusted as we learn more). The Manny Manifolds engine can now be implemented with far less ambiguity, ensuring all contributors follow the same playbook for energy, distance, curvature, motifs, and sandbox commits – solidifying the architecture for consistent evolution.

Sources: The definitions and formulas above are grounded in the Manny Manifolds documentation [8] [31] [4] and analogous principles from cognitive science (predictive coding [5], Hebbian learning [23], hierarchical search in vector spaces [14]), as cited. Each choice was made to satisfy the core design principles: *data as space, conversation as motion, learning as curvature* – now in a rigorously defined form.

---

[1] [2] [3] [4] [6] [7] [8] [9] [10] [11] [12] [13] [15] [16] [17] [18] [19] [22] [23] [24] [25] [27] [28] [29] [33] [35] [40] [41] [42] [43] [44] Manny_Manifolds__Complete_Documentation_Export.pdf
file://file-ArPc98bEY4CdEBhzZzY2Gn

[5] Free energy principle - Wikipedia
https://en.wikipedia.org/wiki/Free_energy_principle

[14] Hierarchical navigable small world - Wikipedia
https://en.wikipedia.org/wiki/Hierarchical_navigable_small_world

[20] [21] [30] MM Cross-Disciplinary Analogies for Gravity & Lens Overlays.md
file://file-P8YxSxc7rVpc7cySoucNyV

[26] Modeling the Dynamic Interaction of Hebbian and Homeostatic ...
https://www.cell.com/neuron/fulltext/S0896-6273(14)00894-0

[31] [32] [34] [36] [37] [38] [39] Manny_Manifolds__Complete_Documentation_Export.pdf
file://file-2R4dCBg1bfufkrK3gqvbpX