Enhancing the Minimal Viable Cognitive Core

July 17, 2025

Quick note on the citation tags you're seeing (e.g., (MVCC p.1)[FN3]): A structured explanation is included at the end in Appendix A: How to Read Filecite Markers. Jump there anytime if you need clarity.

Fast Path to Validating Energy Evaluation with Organ-Based Agents & Hierarchical Memory

1 Introduction: The COA Energy Model as a Unified Control System

The energy model is the circulatory system of the Cognitive Organism Architecture (COA): every signal that matters—collaboration success, coordination overhead, exploration diversity, representational sprawl, memory pressure—flows into a single scalar objective *energy*. Lower is better. Higher indicates friction, waste, or risk.

Rather than bolt on dozens of ad-hoc KPIs, COA collapses them into one mathematically coherent Unified Energy Function. Each subsystem exposes state that maps bijectively onto one term in that function; each control loop acts (approximately) as a contraction mapping on the global energy landscape. When all loops cooperate to descend energy, the organism becomes:

- **Composable:** New subsystems just add new terms (or weight existing ones) without destabilizing others.
- **Auditable:** Every state change is traceable to an energy delta.
- **Provably Stable:** If each loop yields non-positive expected ΔE and weights keep the function bounded below, aggregate behavior converges. (Sketch in §3.x.5.)
- **Adaptive at Scale:** Agents, roles, and memory compete/coordinate through energy; large swarms remain controllable.

Guiding Principle: Every material decision in COA must be expressible as "choose the action expected to reduce global energy the most, subject to constraints." When in doubt, ask: What's the energy delta?

1.1 Energy Descent & Safety Intuition

If every local decision module (agent selector, role optimizer, memory compressor) is biased toward energy descent, you get an emergent Lyapunov-like guarantee: the system resists runaway growth, collapses failure modes faster, and self-regularizes complexity. This is the safety lever that lets a "swarm of swarms" behave like a coherent organism in production.

Goal: Extend your Minimal Viable Cognitive Core (MVCC) so you can quantitatively validate the Unified Energy formalism across (a) agents living inside typed organs and (b) a hierarchical / Holon Memory stack. The emphasis is on a tight experiment loop that lets you observe ΔEnergy as tasks route, agents act, graph state updates, and memory pressure fluctuates.

This doc maps the essentials from the full Cognitive Organism (COA) paper into the smallest incremental build you can reasonably ship to generate hard evidence that the energy layer works as designed.

2 What You Already Have in MVCC

Your minimal note sensibly anchors on a stripped-down loop: Ray Agent → Task → DGL Graph Update → Energy Evaluation → Feedback and stresses auditability as the foundation for a "cognitive control substrate." (MVCC p.1)[FN1] It also frames a layered organism architecture—Coordination, Memory, Reasoning, Energy, Governance—where each layer exposes a state vector, energy contribution, and gradient/feedback signal, and recommends energy minimization + evolvable hypergraphs/roles. (MVCC p.1)[FN2] (MVCC p.1)[FN3]

We'll preserve that loop, add organ boundaries, agent role state, and a 4-tier memory placeholder so Energy can be measured across all contributors.

3 Minimal Validation Questions to Answer

To know the Energy layer is "real," instrument for these falsifiable checks:

Hypothesis	Observable	Expected Signal	Pass Criterio
Stable agent collaboration lowers pairwise energy term	Increase coordination success among same-agent pairs	Downward trend in pair term portion of E	Slope < 0 over
Hypergraph (multi-organ) decomposition improves cost	After escalation, reuse learned pattern	Reduced hyper term share over repeats	>X% drop by
Role diversity prevents overfitting	Collapse of role entropy increases failure	Energy spikes in reg + pair terms	Restoring diver
Memory compression vs staleness trade	Increase CostVQ when compressing	mem term up; stale latency down (or vice versa)	Parameter swe Pareto front

Table 1: These map directly to the Unified Energy formulation where pair, hyper, entropy, reg, and mem terms are explicit. (COA Eq.2 overview p.~5)[FN4]

4 Unified Minimal State Vector

The full COA defines a 4-level hierarchical state: agent, organ, system, memory. (COA Unified State §3.1 p.~5)[FN5] Mirror this at reduced dimensionality:

```
State = {
  'agent': {agent_id: {'h': emb, 'p': role_probs, 'c': capability}},
  'organ': {organ_id: {'h': emb, 'P': agg_role_dist, 'metrics': ...}},
  'system': {'h_hgnn': emb, 'E_patterns': hyperedges, 'mode_w': vec},
  'memory': {'Ma': stats, 'Mw': stats, 'Mlt': stats, 'Mfb': stats}
}
```

Each organ-hosted agent's capability and memory utility should be tracked for lifecycle transitions and later coupling to Energy. The COA lifecycle explicitly links performance + memory utility to capability updates and role shifts. (COA Agent Lifecycle §3.2 p.~6)[FN6]

4.1 Capability & Memory Utility Tracking

Why track both? Capability tells you what an agent *can do* reliably; memory utility tells you how much that agent contributes to collective memory value (retrieval hits, novelty, compression savings). Lifecycle transitions—promotion to `Employed`, demotion to `Scout`, specialization, or retirement—should be triggered by joint thresholds over these two signals rather than performance alone. This mirrors the COA lifecycle rule linking performance + memory utility to role shifts. (COA Agent Lifecycle §3.2 p.~6)[FN6]

Key Metrics per Agent (rolling windows):

- `success_rate`: fraction of tasks completed w/in SLA.
- `quality_score`: graded output (LLM eval or human rating).
- `capability_score` c_i: smoothed aggregation of success & quality.
- `mem_util` u_i: weighted sum of (writes_used_in_future_tasks, hit_rate, compression_gain, salience_flagged_events
- `load`: recent task volume.

Derived Lifecycle Signals:

- `promote_threshold`: if $c_i \geq \tau_c$ and $u_i \geq \tau_u \rightarrow$ candidate `Employed`.
- `scout_threshold`: if c_i low but u_i high in novelty discovery $\rightarrow$ `Scout`.
- `retire_threshold`: persistently low both $\rightarrow$ retire or archive embedding.

Update Equations (EWMA form):

$$c_i = (1 - \eta_c) \cdot c_i + \eta_c \cdot (w_s \cdot \text{success} + w_q \cdot \text{quality})$$
$$u_i = (1 - \eta_u) \cdot u_i + \eta_u \cdot (w_{\text{hit}} \cdot \text{mem\_hits} + w_{\text{compr}} \cdot \text{compr\_gain} + w_{\text{sal}} \cdot \text{salience})$$

where η_* are smoothing factors; weights normalize metrics.

Coupling to Energy:

- w_{ij} pair weights can be scaled by $\min(c_i, c_j)$.
- Mem term `CostVQ` can be discounted when writes come from high-u_i agents (their memory is more valuable).
- Entropy term can bias toward sampling `Scout` agents when global diversity drops.

Lifecycle State Machine (minimal):

- `Scout` $\xrightarrow{(c_i \uparrow \,\&\, u_i \text{ stable})}$ `Employed`
- `Employed` $\xrightarrow{(c_i \downarrow \text{ or } u_i \downarrow)}$ `Scout`
- `Employed` $\xrightarrow{(\text{high } c_i \,\&\, \text{high } u_i \,\&\, \text{specialization pattern})}$ `Specialist(OrganSplit)`
- `Any` –(`retire_threshold`)⟶ `Archived`

Transitions generate events consumed by the Energy evaluator so pair/hyper weights and role distributions update consistently.

4.2 Evolution Axes (Specialization, Accuracy, Smartness, Reasoning)

These are the human-visible fronts on which the organism should improve. Each axis already lives in COA telemetry; we surface and wire them explicitly so evolution feeds back into Energy. **Front**

Axis	Human Meaning	Primary Signals	Energy Terms	Control Loops
Specialization	Agents/organs narrow to niches where they excel	Rising capability within cluster; task KL↓	Pair↓, Reg stable	Slow Loop role PSO; Lifecycle promote/split.
Accuracy	Higher task success/quality	`success_rate`↑, `quality`↑	Pair↓, Hyper↓ (when multi-organ)	Fast Loop agent select; weight updates.
Smartness	Better result per unit cost	E/Task↓; latency↓; mem tradeoffs	Total↓; Mem↔; Reg↔	Adaptive Loop mem; global weight tuning.
Reasoning Depth	Correct multi-step cross-organ plans	`hyperedge_reuse`↑	Hyper↓; Entropy floor	Escalation/hyper learner; entropy guard.

Table 2: *
(COA Eq.2 overview p.~5)[FN4] (COA Agent Lifecycle §3.2 p.~6)[FN6] (COA /energy/gradient API p.~6)[FN8] (COA Hypergraph Decomposition Scenario §6.5 p.~17)[FN18]

Property Vector: $F = [\text{spec}, \text{acc}, \text{smart}, \text{reason}]$ normalized $[0, 1]$. Track windowed deltas; feed them into adaptive weight shaping (see §19).

5 Minimal Energy Function (E_{core})

Below is an enhanced, working-engineer view of the Unified Energy. Use this operational form to wire telemetry and drive the three control loops. Start with a compressed instantiation of Eq. (2):

$$E_{\text{core}}(s) = -\sum_{(i,j)\in\text{organ}} w_{ij} \cdot \text{sim}(h_i, h_j) - \sum_{e\in E} w_e \cdot \text{prod}(g_o \in e) - \alpha H(p) + \lambda_{\text{reg}} \|s\|_2^2 + \beta_{\text{mem}} \text{Cost}_{\text{VQ}}(M)$$

The full formulation—and its interpretation of each cost/benefit term (pair, hyperedge, entropy, regularization, memory cost)—is given in the COA paper. (COA Eq.2 overview p.~5)[FN4] Every subsystem acts on the gradient of this unified energy: local GNNs pick agents via $\nabla_{\text{agent}}E$; PSO updates roles via $\nabla_{\text{role}}E$; the memory controller adapts compression via $\partial E/\partial \text{Cost}_{\text{VQ}}$. (COA Energy Gradient §3.1.3 p.~6)[FN7]

Telemetry Interface Expose `/energy/gradient` returning a JSON breakdown per term so you can watch the energy budget move in real time—this is called out explicitly in the COA design. (COA /energy/gradient API p.~6)[FN8]

5.1 Detailed Algorithms & Provability Notes

This section operationalizes E_{core} for your MVCC build: data structures, incremental updates, gradients for control loops, and proof hooks.

5.1.1 Term Glossary & Instrumentation Table

Below: each energy term, what it means, the minimum telemetry you must log, when to update, and implementation notes.

Term	Operational Meaning	Minimal Stats Needed	Update Event
Pair	Collab quality/affinity of agent pairs (within organ)	`success_count`, `trial_count`, `sim(h_i,h_j)`, `task_type`	After multi-agent task completes
Hyper	Quality/reliability of multi-organ decomposition patterns	`hyper_id`, success, `involved_orgs`, latency	After escalated plan completes
Entropy	Diversity of role allocation across agents in an organ	`role_counts` per agent (E,S,O), p_i	On role update or window close
Reg	Magnitude/complexity of learned state	$\|h\|_2^2$	On state update
Mem	Memory pressure & info loss	`bytes_used`, `recon_loss`, `hit_rate`, `stale_s`	Mem write/evict/ compress

Table 3: *
(COA Eq.2 overview p.~5)[FN4] (COA Energy Gradient §3.1.3 p.~6)[FN7] (COA /energy/gradient API p.~6)[FN8]

5.1.2 Minimal Data Structures & Online Ledgers

Use lightweight rolling ledgers to avoid $O(N^2)$ recompute of energy each tick.

```
class EnergyLedger:
    def __init__(self):
        self.pair = self.hyper = self.entropy = 0.0
        self.reg  = self.mem   = 0.0
    def total(self):
        return self.pair + self.hyper + self.entropy + self.reg + self.mem
```

Pair cache, hyper cache, role dist cache, mem stats sketches follow in §3.x.3.

5.1.3 Incremental Update Routines (Event $\rightarrow \Delta E$)

Emit typed events; update ledgers incrementally:

```
def on_pair_success(ev, ledger, pair_stats):
    ps = pair_stats[ev.pair] # (i,j)
    ps['w'] = ema(ps['w'], ev.success)
    ps['sim'] = ev.sim
    ledger.pair += - ps['w'] * ps['sim']

def on_hyper_exec(ev, ledger, hyper_stats):
    hs = hyper_stats[ev.hyper_id]
    hs['w'] = beta_update(hs['w'], ev.success)
    ledger.hyper += - hs['w'] * ev.goal_prod

def on_role_update(ev, ledger):
    ledger.entropy += -alpha * (ev.H_new - ev.H_prev)

def on_state_update(ev, ledger):
    ledger.reg += lambda_reg * (ev.norm2_new - ev.norm2_old)

def on_mem_event(ev, ledger):
    ledger.mem += beta_mem * ev.cost_delta
```

(COA /energy/gradient API p.~6)[FN8]

5.1.4 Gradient Proxies for Control Loops

Full analytical gradients are optional; MVCC can use proxies that preserve directionality.

- **Fast Loop (Agent Select):** approximate $-\nabla_{\text{agent}}E$ with expected $\Delta_{\text{pair}} + \Delta_{\text{hyper}}$ + entropy bonus.

$$\text{score}_i = w_{\text{pair}} \cdot E_{\text{pair\_delta}}(i,t) + w_{\text{hyper}} \cdot E_{\text{hyper\_delta}}(i,t) - w_{\text{explore}} \cdot \text{entropy\_gain}(i)$$

$$\text{agent} = \arg\min_i(\text{score}_i)$$

  (COA Energy Gradient Agent Select p.~6)[FN7]

- **Slow Loop (Role PSO):** particles = candidate role distributions; fitness = windowed E. (COA Agent Lifecycle §3.2 p.~6)[FN6]
- **Adaptive Mem Loop:** sweep compression knob k; estimate slope dE/dk. (COA /energy/-gradient API p.~6)[FN8]

5.1.5 Provability Hooks (Sketch)

- **Bounded Below:** finite $|w|$; reg, mem $\geq 0 \Rightarrow E$ bounded. (COA Eq.2 overview p.~5)[FN4]
- **Energy-Contraction:** each loop produces action with $E_{\text{expected\_post}} \leq E_{\text{pre}}$. Robbins–Siegmund $\Rightarrow$ convergence. (COA Energy Gradient §3.1.3 p.~6)[FN7]
- **Entropy Floor:** ensure $H(p) \geq h_{\min}$ to avoid local minima lock-in. (COA Agent Lifecycle §3.2 p.~6)[FN6]
- **Mem Guard:** β_{mem} schedule prevents divergence under load. (COA /energy/gradient API p.~6)[FN8]

5.1.6 Telemetry Payload Extension

Augment `/energy/gradient` with per-task deltas + rolling slopes so you can correlate front-axis metrics (§2.2) with energy descent.

```
{
  "ts": 1721184000,
  "E_terms": {"pair":0.12,"hyper":0.04,"entropy":-0.02,"reg":0.01,"mem":0.03,"
    total":0.18},
  "deltaE_last": -0.005,
  "window": {"W":100,"slope":-1.2e-3},
  "agent_grad_proxy": {"a7":-0.03},
  "mem_pressure": {"Mw":0.62,"Mlt":0.41}
}
```

(COA /energy/gradient API p.~6)[FN8]

6 Agents Living in Organs: Minimal Execution Model

COA models a swarm-of-swarms in which agents cluster into organ-level swarms (e.g., Cognitive, Actuator, Utility) coordinated globally via an OCPS-gated router and locally via 200ms GNN loops; memory augments execution. (COA OCPS Fast Router §1 Intro p.˜1)[FN9] Adopt the smallest viable subset: define 2–3 organs, each a Ray actor, with typed task schemas.

Sample Organ Interface

Each organ exposes a typed interface (tasks accepted, response schema, periodic state summary) to support fast-path routing and predictable energy accounting. The COA highlights standardized organ interfaces as the basis for scalable routing and reduced task entropy. (COA Standardized Organ Interface §5.1 p.˜14)[FN10]

Routing Flow

Maintain a routing table (fast path) and escalate to a hypergraph reasoner when patterns are novel—COA's coordination structure uses exactly this fast vs HGNN split with ˜90% fast path. (COA Fast vs HGNN Split §3.2 p.˜6)[FN11]

7 Agent Lifecycle Hooks You Need

Instrument capability growth from (`success_rate`, `memory_utility`) and role transitions from `Employed`→`Scout` once capability crosses a threshold; scouts discovering reusable patterns can trigger organ fission or specialization. This lifecycle is spelled out in COA's agent evolution model. (COA Agent Lifecycle §3.2 p.˜6)[FN6]

8 Three Control Loops per Organ (Minimum Two to Start)

The COA runs a Three-Loop Agent Fabric: fast reactive (Local GNN, ˜200ms), slow adaptive (PSO, ˜2s), and an optional hourly flywheel optimization—each driven by the global energy landscape and contractive by design. (COA Three-Loop Fabric §7 p.˜17)[FN12] For MVCC validation, implement the first two; stub the third with a no-op telemetry hook.

9 Hierarchical / Holon Memory Mini-Stack

The COA memory hierarchy uses tiers M_a (agent), M_w (working/organ-local), M_{lt} (long-term), and M_{fb} (flashbulb / salient incidents), bundled into a Holon Memory Fabric with freshness bound $\Delta t_{\text{stale}} \leq 3$s in the formal model. The production-oriented section further expands this into a unified HMF with meta-learning control to reduce staleness. (COA Holon Memory Upgrade §8 p.˜21)[FN13]

9.1 Minimal Tier Definitions

M_a: per-agent scratch (vector emb history, capability stats); maps to Tier 0 in COA. (COA Memory Tier0 §8.1 p.˜21)[FN14]

M_w: organ-local working set; prefetch / broadcast reduce access cost, as described in memory-augmented agent execution. (COA Working Memory Ops §5/8 p.~21)[FN15]

M_{lt}: persistent cross-organ holon store; target for consolidation & reuse; supports energy hyperedge learning. (Context from same memory section above.) (COA Working Memory Ops §5/8 p.~21)[FN15]

M_{fb}: time-bounded salient incident log; used to spike mem term + accelerate consolidation; see adaptive compression + salience gating. (COA Adaptive Compression §8 p.~21)[FN16]

9.2 Compression Control & CostVQ

Adaptive compression (VQ-VAE thresholding under memory pressure) directly feeds the mem energy term; vary the threshold to see tradeoffs.

9.3 Freshness & Improvement Bound

With adaptive replay + compression scheduling the COA tightens expected staleness (E[stale])—use this as an evaluation target when you wire telemetry into M_w/M_{lt} writes. (COA Freshness Bound §6.10 p.~19)[FN17]

10 Minimal Data Model & Metrics Schema

Below is a thin schema for logging per-task energy contributions:

```
{
  "task_id": "uuid",
  "organ": "cognitive",
  "agents": ["a7","a12"],
  "path": "fast" | "esc",
  "energy": {
    "pair": 0.12,
    "hyper": 0.04,
    "entropy": -0.02,
    "reg": 0.01,
    "mem": 0.03,
    "total": 0.18
  },
  "memory_stats": {"mw_hits":3,"mlt_hits":1,"compr_ratio":1.8},
  "roles": {"a7": {"E":0.7,"S":0.2,"O":0.1}},
  "capability": {"a7":0.55},
  "latency_ms": 42
}
```

The `/energy/gradient` endpoint in COA returns a per-term JSON so operators can debug, which motivates this schema. (COA /energy/gradient API p.~6)[FN8]

11 Execution Pipeline v2 (Energy-Aware)

Current MVCC: Ray → Task → DGL → Energy → Feedback.
Enhanced:

1. **Ingress:** Stimulus arrives; OCPS-lite drift test decides fast vs escalate. (COA global coordination fast/HGNN split.) (COA Fast vs HGNN Split §3.2 p.~6)[FN11]

2. **Fast Path:** Route to target organ actor; local GNN selects agent using $\nabla_{\text{agent}} E$. (COA Energy Gradient Agent Select p.~6)[FN7]

3. **Escalation Path:** Hypergraph pattern selection across organs; update hyper term weights. (COA hyperedge escalation.) (COA Hypergraph Decomposition Scenario §6.5 p.~17)[FN18]

4. **Agent Exec:** Agent acts; log success + memory utility for lifecycle updates. (COA Agent Lifecycle §3.2 p.~6)[FN6]

5. **Memory Writes:** Working-set update + optional broadcast; drive mem cost + freshness metrics. (COA Working Memory Ops §5/8 p.~21)[FN15]

6. **Energy Eval:** Compute E_{core} delta per task; stream JSON to telemetry endpoint. (COA /energy/gradient API p.~6)[FN8]

12 Minimal Algorithms

OCPS-Lite Drift Gate Use a rolling KL divergence or CUSUM on task embedding distribution; if $> \tau$, escalate. Full COA uses an OCPS drift detector gating escalation probability p_{esc} in stability bounds. (COA OCPS Fast Router §1 Intro p.~1)[FN9]

Local Agent Selector Score = weighted sum of (capability, accuracy, availability, memory-recall, cache-coverage) → pick argmin Energy proxy. This mirrors the energy-driven suitability score in COA. (COA Suitability Factors §7 p.~17)[FN19]

Capability Update $c_i += \alpha \cdot \text{success\_rate} + \beta \cdot \text{memory\_utility}$; check threshold for role shift. Directly lifted from COA lifecycle rule. (COA Agent Lifecycle §3.2 p.~6)[FN6]

Memory Compression Schedule If M_w pressure > target, raise compression gate (VQ-VAE) and attribute added CostVQ into mem term. Based on COA adaptive compression control.

13 Experimental Harness

Run synthetic workloads to stress each energy term:

Exp A – Pairwise Collaboration Learning Generate tasks that require 2 agents in same organ; gradually bias router toward successful pairs; expect falling pair energy. Uses pairwise gain term from Eq. (2).

Exp B – Hyperedge Reuse Inject rare multi-organ tasks → escalate → learn hyperedge → reuse; hyper term should drop as w_e strengthens. Eq. (2) hyperedge component.

Exp C – Role Entropy Ablation Lock roles (collapse p_i) vs allow diversity; entropy term demonstrates exploration value. Eq. (2) entropy component.

Exp D – Memory Pressure Sweep Throttle compression threshold; watch mem term vs freshness; target Δt_{stale} and improvement bounds from COA. (COA Freshness Bound §6.10 p.~19)[FN17]

14 Dashboards & Developer Tooling

Your MVCC memo already pushes for CLI + dashboards (run-task, visualize-energy, DGL metrics). Extend to show per-term spark lines, organ hit rates, and memory compression levels. (MVCC Tooling p.2)[FN20] Recommended Tech picks (Ray actors, lightweight LLM agents, PGVector+Neo4j HMF, VQ-VAE compression, RLlib meta-controller) are spelled out in the COA implementation insights—use these defaults to cut decision time. (COA Tech Stack §6.12 p.~19)[FN21]

15 Phased Build Checklist

Phase 1 (Day 1–3): Core Loop Instrumented

- Implement `/energy/gradient` returning per-term JSON. (COA /energy/gradient API p.~6)[FN8]
- Ray organ actors + agent registry. (Organs as scalable units in COA.) (COA OCPS Fast Router §1 Intro p.~1)[FN9]

Phase 2 (Day 4–7): Lifecycle & Memory Hooks

- Capability + `memory_utility` update rule. (COA Agent Lifecycle §3.2 p.~6)[FN6]
- Working-memory prefetch & broadcast stub. (COA Working Memory Ops §5/8 p.~21)[FN15]

Phase 3 (Day 8–12): Energy Experiments

- Hyperedge escalation path. (COA Hypergraph Decomposition Scenario §6.5 p.~17)[FN18]
- Adaptive compression knob + mem cost telemetry.

Phase 4 (Day 13+): Performance & Freshness

- Measure staleness vs compression; aim toward $\Delta t_{\text{stale}} < 3$s baseline, improved with adaptive scheduling. (COA Freshness Bound §6.10 p.~19)[FN17]

16 Minimal Data Structures (Type Hints)

Feel free to copy/paste into your repo scaffold.

```python
from dataclasses import dataclass, field
from typing import Dict, List, Literal, Tuple
import numpy as np

Role = Literal['E','S','O']  # Employed, Scout, Onlooker
Path = Literal['fast','esc']

@dataclass
class AgentState:
    h: np.ndarray           # embedding
    p: Dict[Role, float]    # role probabilities
    c: float                # capability
    mem_util: float = 0.0   # contribution to shared memory value metric

@dataclass
class OrganState:
    h: np.ndarray
    P: Dict[Role, float]    # aggregate role distribution
    metrics: Dict[str, float] = field(default_factory=dict)

@dataclass
```

```
class MemoryStats:
    Ma: int; Mw: int; Mlt: int; Mfb: int
    compr_ratio: float = 1.0
    stale_s: float = 0.0

@dataclass
class EnergyTerms:
    pair: float; hyper: float; entropy: float; reg: float; mem: float
    total: float
```

(Connect fields to energy terms per Eq. (2).) (COA Eq.2 overview p.~5)[FN4]

17 Quickstart Pseudocode: Task Execution Cycle

```
def execute_task(task):
    if drift_detect(task):
        path = 'esc'
        organs = hgnn_decompose(task)     # hyperedge pattern
    else:
        path = 'fast'
        organs = [router_lookup(task)]

    energy_before = energy_read()

    for organ in organs:
        agent = organ.select_agent(task)  # uses grad(agent,E) proxy
        result = agent.run(task)
        organ.update_metrics(result)
        memory_write(organ, result)       # Mw; may escalate to Mlt/Mfb

    graph_update(task, result)
    energy_after = energy_read()
    log_delta_energy(task, path, energy_before, energy_after)
```

Fast vs escalation gating + agent selection on energy gradient replicate COA control flow. (COA Energy Gradient Agent Select p.~6)[FN7] (COA Hypergraph Decomposition Scenario §6.5 p.~17)[FN18]

18 Validation Scorecard Template

Track runs in a spreadsheet or dashboard against target signals:

Run	**Pair ↓**	**Hyper ↓**	**Entropy maintained**	**Mem Cost vs Stale**	p_{fast}
A1	✓	–	✓	–	0.91
B1	–	✓	✓	–	0.88
D1	–	–	–	Tradeoff curve	0.90

Table 4: *
Fast-path dominance ~90% is a design invariant in COA's coordination model; use it as a regression guard. (COA Fast vs HGNN Split §3.2 p.~6)[FN11]

19 Next Extensions

Once core validation succeeds:

- Add suitability score factors (capability, accuracy, availability, memory recall) to local GNN selection to further align with COA's formulation. (COA Suitability Factors §7 p.~17)[FN19]
- Instrument dashboard energy drill-down CLI hooks as recommended in MVCC tooling guidance. (MVCC Tooling p.2)[FN20]
- Scale out tech picks from COA's implementation mapping (Ray actors, PGVector/Neo4j, RLlib meta-controller). (COA Tech Stack §6.12 p.~19)[FN21]

20 Wrap

By grafting just a few high-leverage concepts from the full Cognitive Organism—hierarchical state vector, unified energy gradient API, organ-scoped agents with lifecycle rules, and a thin Holon Memory—you can see the energy landscape move under controlled experiments within days. That instrumented evidence will derisk deeper auto-evolution, governance, and large-scale swarm behaviors later.

21 Adaptive Energy Shaping (Closing the Loop from Front Metrics)

Below we define the forward + backward coupling that lets improvements in Specialization, Accuracy, Smartness (adaptive efficiency), and Reasoning Depth actively reshape the Unified Energy landscape while preserving stability.

21.1 Windowed Metrics → Weight Adaptation

Every W tasks compute normalized deltas: $\Delta_{\text{spec}}, \Delta_{\text{acc}}, \Delta_{\text{smart}}, \Delta_{\text{reason}}$. Update energy weights within safe bounds:

$$\begin{aligned}
w_{\text{pair}}* &= (1 + \gamma_{\text{spec}}\Delta_{\text{spec}} + \gamma_{\text{acc}}\Delta_{\text{acc}})\\
w_{\text{hyper}}* &= (1 + \gamma_{\text{reason}}\Delta_{\text{reason}})\\
\alpha* &= (1 - \gamma_{\text{spec}}\Delta_{\text{spec}} + \gamma_{\text{reason}} \cdot \text{entropy\_floor\_boost})\\
\lambda_{\text{reg}}* &= (1 + \gamma_{\text{smart}} \cdot \max(0, \text{reg\_growth}))\\
\beta_{\text{mem}}* &= (1 + \gamma_{\text{smart}} \cdot \text{mem\_pressure})\\
&\text{project\_bounds}()
\end{aligned}$$

(COA Eq.2 overview p.~5)[FN4] (COA Energy Gradient §3.1.3 p.~6)[FN7]

21.2 Evolution Operators

Triggered structural actions that reconfigure the organism when sustained energy signals align with front metrics:

Operator	**Trigger**	**Energy Rationale**	**Action**
Agent Specialization Split	High c_i in narrow task cluster; pair $\downarrow$	Make implicit specialization explicit; may reduce reg	Fork Specialist; adjust routing priors.
Organ Fission	Low cross-sim among agents; noisy hyper	Partition to sharpen pair gradients	Create new organ; migrate agents.
Scout Injection	Entropy $< h_{\text{min}}$; hyper stalls	Increase exploration to drop future hyper	Promote Scout roles; raise α locally.
Memory Promotion Burst	Hyper depends on stale data; mem low	Retain data to strengthen hyper reuse	Relax compression; pin memory spans.

Table 5: *
(COA Agent Lifecycle §3.2 p.~6)[FN6] (COA Hypergraph Decomposition Scenario §6.5 p.~17)[FN18] (COA /energy/gradient API p.~6)[FN8]

21.3 Stability Guards

- **Bounded Weights:** Clamp $[w_{\text{min}}, w_{\text{max}}]$; ensures E bounded below. (COA Eq.2 overview p.~5)[FN4]
- **Entropy Floor:** Maintain $H(p) \geq h_{\text{min}}$ to avoid mode collapse. (COA Agent Lifecycle §3.2 p.~6)[FN6]
- **Non-positive Expected ΔE:** Reject batch weight updates if $\Delta E_{\text{expected}} > \epsilon$. (COA Energy Gradient §3.1.3 p.~6)[FN7]
- **Memory Watermarks:** Auto-raise β_{mem} past high watermark; prevents mem blowup. (COA /energy/gradient API p.~6)[FN8]

21.4 Evolution Tick Skeleton

```
def evolution_tick(window_metrics):
    delta_spec, delta_acc, delta_smart, delta_reason = derive_front_deltas(
   window_metrics)
    adapt_energy_weights(delta_spec, delta_acc, delta_smart, delta_reason)
    run_lifecycle_updates()        # uses c_i & u_i
    maybe_specialize_agents()
    maybe_split_organs()
    maybe_inject_scouts()
    maybe_promote_memory()
    recompute_energy_baseline()
```

A How to Read Filecite Markers

You've asked exactly the right question: What do these look-weird tags mean? Example: `(MVCC p.1)[FN3]`

Let's decode every field:

Important Clarifications

- **Tool-extracted lines, not PDF page lines.** The L# markers come from the plain-text extraction the `file_search` tool produced. A single PDF paragraph may span multiple L# lines depending on extraction width.

Segment	Meaning
`(`	Opening delimiter that starts the citation.
`MVCC p.1`	The file ID and page number.
`)[FN3]`	Closing delimiter that ends the citation.

- **Inclusive Range.** L63-L69 means start at line 63 and include through line 69 (both ends included).
- **Single Range Per Citation.** Each filecite must point to one contiguous range. If you need two disjoint ranges, supply two separate filecite tags.
- **Multiple Citations in One Sentence.** When you see: ...text... (MVCC p.1)[FN1] (MVCC p.1)[FN2], that means the statement is supported by both source ranges.
- **Why So Specific?** Fine-grained line ranges make your research auditable: readers (or downstream tooling) can open the original PDF extract and jump straight to the supporting evidence.

Mapping Back to Your Paper If you want to know where in the PDF a given L# appears:

- Open the source (we can do this together) and I'll locate the lines and report the PDF page + contextual snippet.
- We can build a quick helper script that maps extracted line indices back to PDF page spans using a text locator.

Quick Visual Legend

```
[text I wrote summarizing your doc]   <-- my prose
(MVCC p.1)[FN1]                      <-- evidence: lines 4-23 from that
                                         file's extracted text
```

If you'd like, tell me which citation you'd like to inspect (e.g., L63-L69) and I'll open the underlying file, show you the actual excerpt, and point you to the PDF page.